TESIS DOCTORAL

Contribution to improve mobility uses through context-awareness

Autor:

Elena Yndurain

Director:

Dr. Celeste Campo

INGENIERIA TELEMATICA

TESIS DOCTORAL

CONTRIBUTION TO IMPROVE MOBILITY USES THROUGH CONTEXT-AWARENESS

Autor: Elena Yndurain

Director: Dr. Celeste Campo

Firma del Tribunal Calificador:

Firma

Presidente:

Vocal:

Secretario:

Calificación:

Leganés, 21 de Diciembre de 2012

To my parents, who have taught me to become a person in all contexts of life.

In loving memory of my father

Acknowledgments

There are many people that have helped me in one way or another to accomplish this PhD dissertation and that I would like to give them a very big thank you.

For starters, I would like to thank Celeste Campo, for accepting to be my PhD advisor, for supervising my research work, and helping me transform it into a thesis dissertation.

Fernando Corbacho, for explaining me the machine learning underlying concepts, for his guidance and his constant encouragement, as well as Claudio Feijóo for his support and helping me to get started in writing JCR journal papers.

To the Bing Team: Jordi Ribas for encouraging me to do a PhD and a research internship and opening the doors of Microsoft's European Search Technology Center, Mike Miles for giving me the opportunity to do my research internship on his team, Daniel Bernhard for our discussions on context-awareness and search, and working with me in publishing an article on the topic, Marcus Duyzend for helping me get the necessary infrastructure to run my algorithms, and Jose Santos for his invaluable discussions on how classifiers work.

The researchers at the Universidad Carlos III from the Nokia Chair led by Carlos Delgado Kloos, for sparking my interest on research while we worked together, and to those researchers that also helped me with different requests throughout my thesis: when looking for research internships, coding questions, discussions on the dissertation and moral support on PhD life (Javier Carbó, MariLuz Congosto, Jose Félix Kukielka, Mari Carmen Panadero, Alicia Rodriguez, Carlos García Rubio, and Tomás de la Rosa).

I would also like to thank my family and friends that I have dragged into proofreading my articles and even this dissertation while I was closing it up: my mother, my uncle Félix and my friends Fernando, Osama, Sarah, Scott, and Steve.

The researchers that volunteered to collaborate on reviewing my dissertation, Peter Bailey, Sergio Gutierrez, Ora Lassila, and Fabio Paternò, for their time, their valuable discussions and feedback on my research.

Last, but not least, I would like to thank my friends and family for their patience putting up with my stress and constant rejections to socialize because I was busy working on my thesis. Thank you for your understanding and your support throughout these years.

Resumen

Describe Dey, en su artículo “Towards a Better Understanding of Context and Context-Awareness” cómo la percepción del contexto (context-awareness) cobra importancia en las aplicaciones en las que el contexto del usuario cambia con rapidez, como es el caso en los entornos móviles de la computación ubicua. Dey, en su artículo, define contexto como “cualquier información que pueda usarse para caracterizar la situación de una entidad”. En entornos móviles, dicha entidad es el dispositivo móvil en sí mismo. Este aparato, al ser ubicuo y centrado en las personas, puede captar continuamente información tanto de los usuarios como de su contexto a través de sus sensores.

El uso del contexto ha cobrado importancia en entornos de computación ubicua desde la década de los 90, y esta técnica se ha empleado en dispositivos móviles para mejorar su utilización y aplicación. Para que el área de percepción de contexto se convierta en una realidad, se necesita más investigación, sobre todo en el área de predicción de contexto que amplíe las posibilidades de las aplicaciones que usan información de su contexto. En esta tesis doctoral, nos centramos en el uso de los datos obtenidos de los sensores del móvil y en el comportamiento del usuario, para deducir el contexto presente predecir el contexto futuro, mejorando así la usabilidad del móvil y las funcionalidades de sus aplicaciones. Contribuimos a la computación de percepción del contexto móvil demostrando cómo los dispositivos móviles pueden aprender automáticamente sobre el contexto en el que está el usuario y adaptarse al mismo para mejorar la experiencia de movilidad.

Comenzamos nuestro trabajo realizando un estudio del estado del arte de propuestas de percepción de contexto para sistemas y aplicaciones móviles, así como de las herramientas para intuir el contexto a partir de variables existentes del entorno. Analizamos las carencias que tienen en su aplicación al área de la movilidad y hacemos propuestas de cómo resolverlas a lo largo de la tesis.

Primero sentamos las bases de la tesis definiendo el concepto de percepción de contexto (“context-awarenes”) y realizamos una propuesta de arquitectura de derivación del contexto actual y predicción del contexto futuro desde un punto de vista de un entorno móvil.

Existen muchas definiciones de contexto, percepción de contexto y arquitecturas, pero hay pocas orientadas exclusivamente a movilidad. Además todas se centran en la derivación del contexto actual en vez de hacerlo en la predicción del contexto futuro. Desarrollamos un modelo que nos permite captar, procesar y unificar variables de fuentes heterogéneas para que puedan ser utilizadas por el algoritmo de aprendizaje automático para intuir y predecir contexto. También probamos y referenciamos varios algoritmos de aprendizaje automático para poder recomendar los algoritmos que consideramos más apropiados para intuir

contexto en entornos de movilidad. Hacemos una propuesta de mejora en la que combinamos los algoritmos de predicción en línea con los algoritmos de clasificación para poder así predecir el contexto futuro además del contexto actual intuido por el clasificador.

Evaluamos nuestra propuesta con datos reales de uso del móvil disponibles en el proyecto "Reality Mining", en el cual se captan datos de uso diario de móviles de aproximadamente 100 Smartphones Nokia usados por estudiantes universitarios durante un año académico. Finalmente concluimos dando un ejemplo de cómo aplicar nuestra arquitectura y el modelo propuesto demostrando como enriquece la experiencia de búsqueda en un dispositivo móvil el hecho de incluir un módulo de percepción de contexto en los buscadores móviles. Usamos el buscador Bing para todos los ejemplos de búsquedas.

Abstract

Dey, in his paper "Towards a Better Understanding of Context and Context-Awareness", argues that context-awareness is important in applications in which the user's context changes rapidly, such as in mobile environments for ubiquitous computing. In his paper, Dey defines context as "any information that can be used to characterize the situation of an entity". In mobile environments, the entity is the mobile device itself. The device is both pervasive and person-centric; it can continuously capture information about its users and their context through its sensors.

The use of context has gained importance in ubiquitous computing since the 1990s, and the technique has recently been used in mobile devices to improve their uses and applications. For mobile context-awareness to become a reality, further research is required, particularly in the field of context prediction, which can expand the possibilities of context-awareness applications by expanding the applications' situation awareness.

In this PhD dissertation, we focus on the use of data obtained through mobile device sensors and user behavior to derive and predict context to improve mobility for both the users' experience and for the applications' functionality. We contribute to context-aware mobile computing by showing how mobile devices can automatically learn from the user's context and can adapt to improve the mobile experience.

We begin our work with a state-of-the-art analysis of "context-awareness" proposals for mobile systems and applications and of the current tools used to infer context from the existing environmental variables. In this dissertation, we analyze the existing gaps in mobile environments and propose solutions to resolve these issues.

We first define "context-awareness" and propose an architecture to predict context from a mobility perspective. Numerous definitions of context, context-awareness and architectures exist, but few focus exclusively on mobility. Moreover, all of the definitions are oriented towards context inference rather than towards a prediction of future context. We develop a model that captures, processes and unifies variables from heterogeneous sources for use by a machine-learning algorithm that infers and predicts the context. We also test and benchmark several machine-learning algorithms in our architecture so that we can recommend those algorithms that we consider most appropriate for inferring context in mobility environments. We propose the combination of on-line prediction algorithms and classifier algorithms to enhance context derivation with future context prediction. We evaluate our proposal utilizing real data from the Reality Mining project, which captures data from the daily mobile usage of c.100 Nokia smart phones during an academic year. We conclude with an example of how to apply our proposed architecture and model, and we

demonstrate its enrichment of the search experience with a mobile device by including a “context-awareness” module in mobile search engines. We use Bing as the search engine for all of our search examples.

Index

Index of figures

Index of tables

Index of equations

1 Introduction

Context awareness has gained importance in the field of distributed systems since the 1990s, and it is now a fundamental ingredient of ubiquitous computing applications such as mobile devices because of the pervasive nature of such applications and because the applications focus on the user's life. New smartphones are continuously able to capture context-related information such as location, interaction with the mobile device, and information about the surroundings (i.e., video, audio, noise level, and temperature) through their embedded sensing capabilities (Albrecht Schmidt, Michael Beigl et al. 1999), (Gellersen, Schmidt et al. 2002), (Korpipää and Mäntyjärvi 2003), (Coutaz and Crowley 2005), (Nicholas D. Lane, Emiliano Miluzzo et al. 2010), (Bilton 2011), which allow them to capture contextual data and convert those data into signals that can be understood by applications. Examples of common sensors found in modern smartphones are cameras, GPS receivers, microphones, accelerometers, digital compasses, and network connectivity interfaces, among others.

The use of context is important for interactive applications and particularly for applications where the user's context is rapidly changing, such as in both handheld and ubiquitous computing; if used properly, context information can be used to adapt the mobile apps to each user by augmenting them with real-world information related to the users' profiles and behavioral patterns. In particular, the information can be used to improve mobile usability and the effectiveness of the interaction. The inclusion of context information has sparked an explosion of development of new mobile applications (Grauballe, Perrucci et al. 2008), and many mobile devices already include some simple context functionalities, such as functionalities connected to the inclination or movement of the device or those that adapt mobile apps to location.

Through the use of context, we can change the way we interact virtually with the physical world around us. When people interact with each other, they implicitly use context information; contextualizing is performed unconsciously. However, if we use context information consciously, we can utilize the benefits that context can provide, such as making interactions more effective (Bouquet, Serafini et al. 2008). The beauty of the use of context in mobile applications is that context relates to a wide variety of uses beyond the values supplied by physical sensors (Brown, Bovey et al. 1997).

With the proper understanding of what context is and how it can be used, application developers can determine what features they want to include in their applications (Dey 2001); following this approach in our work, we define context from an end-user point of view. Our research focuses on using the multitude of sensors available in modern smartphones, which are enabling new applications across a wide variety of domains (Nicholas D. Lane, Emiliano Miluzzo et al. 2010) (Bilton 2011). We utilize the sensors to

capture a rich view of the contextual data pertaining to mobile device usage, the user and the user's surroundings.

Sensors provide us with vital signals about the user's context that can be used when developing a mobile app. Our user-centric approach makes us to consider a user's interaction with a mobile device. There are several common variables accessible in mobile phones that can provide relevant information about the user's situation and context, which can affect other mobile interactions.

We focus on how to capture context signals, process them and use them to derive context, how to model context and how to use machine learning algorithms for prediction. Given the current proliferation of sensors and the new contextual information sources that will appear, we must design an architecture that can include new contextual inputs.

There is a wide variety of research related to context prediction and pattern discovery on-line (Sigg, Haseloff et al. 2010), (Song, Kotz et al. 2006; Gopalratnam and Cook 2007), (Kim, Helal et al. 2010) that evaluates the performance of different unsupervised algorithms. All of the current research agrees that the primary requirement for these algorithms is a good balance between prediction accuracy and low processing and memory requirements so that the algorithms are suited to mobile devices and do not compromise performance. In our research, we analyze how well prediction algorithms work for ubiquitous computing environments by testing them with real-world data.

In this dissertation, we present the definition of a model and its underlying architecture to use contextual signals from mobile computing. In particular, this thesis investigates the following hypothesis: *by using a model that combines context variables and an appropriate architecture adapted to a mobile environment, context awareness can improve mobile applications by enriching them.*

1.1 Motivation

Emerging pervasive computing technologies transform the way we live by embedding computation in our environment. To help users avoid being overwhelmed by the complexity of the existing technologies, applications should automatically adapt to the application's changing context, i.e., the physical and computational environment in which the applications run. This concept is the basis of context-aware computing, the goal of which is to reduce the explicit information a user requires during interaction with a computer; the motivations for this reduction are the mobility and activity of users in ubiquitous computing environments (Dargie 2009).

In this thesis, we analyze the whole context-awareness process with the goal of explaining how to use context awareness to improve mobile communications. Our goal is to demonstrate how mobile apps can become more context-aware by adapting them to each

user's circumstances, which leads to an improved user experience. Usability experiences can be improved by defining models that learn automatically from the context and adapt the mobile uses and applications accordingly. In our work, we propose a suitable model to capture heterogeneous context data from many mobile sensors and describe the architecture to support context-aware mobile apps.

1.2 Objectives

In this thesis, we consider several aspects of mobile context awareness and propose a specific model for mobile devices that can profit from the contextual data sensors can capture to contextualize the applications running on the devices.

The principal goals of our research are as follows:

1. To understand the evolution of context awareness and the current state of the art and to present some background about the tools that are used in the research.
2. To provide a thorough definition of context awareness focused on mobility and understanding what benefits context awareness can provide for mobile uses.
3. To propose an underlying light architecture to make mobile applications context-aware; we will explain each of the architecture's modules and the necessary steps to implement the architecture.
4. To develop a framework to combine heterogeneous sensor signals into a coherent context model; we will explain how to capture and process raw data to translate those data into contextual information that can be used by an application.
5. To recommend the classifiers that are most appropriate to be used to infer context in a mobile environment and to propose an improvement that will enable the classifiers to predict future context states as well.
6. To analyze how well prediction algorithms work for ubiquitous computing environments through experimental tests using mobile data available from the Reality Mining project (Eagle, Pentland et al. 2009).
7. To provide a real–world example of mobile usage of context awareness through search engines, to recommend future enhancements and to analyze the limitations of current Internet search engines.

1.3 Work plan

We now describe the work plan that we have followed to accomplish the objectives:

- Analyzed the characteristics of context-aware mobile computing systems, the systems' components, and the agents involved and **to contribute to the analysis of the existing gaps to make these systems available for the mass market**. To accomplish this task, we performed the following subtasks:
  - To analyze the different approaches to define context that have been presented in previous research and to contribute a new mobile-oriented definition.
  - To analyze the existing context-aware applications to understand how they work, what contextual information they possess, how they make use of the information and the benefits the information provides.
  - To analyze contextual data and to identify the nature of contextual information and of the sensors that capture the information to utilize it in applications.
- Analyzed the existing contextual data available in mobile environments and to contribute **by normalizing the data using a context model that can be used to contextualize apps:**
  - To find appropriate real-world mobile datasets that will allow us to test the model by evaluating the existing datasets that capture context-related mobile information.
  - To evaluate and test the different machine learning algorithms used for on-line prediction and classification using the real-world dataset to recommend the most suitable algorithms.
  - To identify the aspects to take into consideration when developing a contextual application for a mobile device, including different mobile platforms, sensors and the limitations they pose for development.
  - To contribute by improving classifiers' performance, thus enabling them to predict future context states by adding an on-line machine learning algorithm to the input data.
- To analyze the context-aware mobile architectures proposed in the literature and those that exist that support the contextual apps analyzed and **to contribute by proposing an architecture that can be included in mobile apps to contextualize them.**
  - To define the necessary steps required to contextualize a mobile application by detailing the specific inputs and outputs that each will have.
  - To define the architecture modules, data flow and functionality designed for a mobile environment and to implement a light version of the different modules of the architecture.
  - To provide a real-world mobile-use example of how to apply the defined model and architecture in a mobile service and to identify the benefits context awareness yields.

1.4 Structure

In this chapter, we introduce the motivations for contextualizing mobile applications and the aim of our thesis.

- Chapter 2 provides the necessary background by exploring the state of the art in context awareness and mobility and by identifying the existing gaps. We describe the core concepts of context awareness, focusing on the mobile environment. We also provide some historical perspectives and trends for context-aware mobile applications.
- Chapter 3 describes the tools that will be used in our experiments, where we select a real-world dataset to use to test our research proposal and the machine learning algorithms that we will run. We define terms and provide an overview of concepts from machine learning and a description of the algorithms we will use in our work. We analyze and describe the existing real life datasets, explaining why we have selected the Reality Mining dataset for our work, and we describe the dataset in depth to understand the type of data it provides for our research.
- Chapter 4 proposes a definition of context awareness in mobile applications and explores the concept in depth. We also describe a model of a mobile context-aware system and analyze the variables it should include and the underlying architecture, thereby explaining the full process required for context awareness.
- Chapter 5 empirically evaluates different types of machine learning classifier algorithms and benchmarks the results when they are used to predict static context states using multivariable analysis. We explore possible irrelevant and redundant features to select the optimal feature set for the proposed classifier model to be used in the proposed architecture.
- Chapter 6 empirically tests the inference of context states dynamically to obtain the future context variables to include in the context vector to allow the classifier to predict future context states. Using a sequence of previous events that can be modeled as a stochastic process, we use a learning algorithm based on Markov models to perform in-depth real-world tests and to analyze proposals to improve the algorithm.
- Chapter 7 illustrates how to apply the proposed model in a real-world example of a mobile use. We choose a mobile search engine for our example and explain what benefits context awareness will bring to the results of a query and how to contextualize mobile queries.

Finally, Chapter 8 presents the conclusions of our work and explains the future work that remains.

1.5 History

On this section we briefly summarize the history behind the thesis creation and specify which are the articles published over the last years and on which part of the research results is reflected.

The idea of researching on context-awareness mobile computing came up while collaborating with Nokia's research center when I realized the great potential that context-awareness had to improve mobility uses, given the wide variety of sensors that were added to mobile devices that were accessible for third party developers through SDKs and APIs.

I started my research by doing an in-depth analysis of the current literature in context-awareness, focusing on mobility. Having done this, I realized there were several gaps in the industry and academics on context-awareness and mobile computing. We reflected these gaps on our article "Context-aware mobile applications: implications and challenges for a new industry" (Yndurain, Feijoo et al. 2010) which we published in 2010 on the Journal of the Institute of Telecommunications Professionals volume 4. On this publication we presented an overview of the state of the art in context-awareness mobile computing analyzing the existing gaps and challenges that needed to be addressed to make it a reality beyond pilots and ad-hoc solutions.

After gaining a thorough understanding of context-awareness and mobile computing, we started modeling mobile context data into different states to use on-line machine learning algorithms to predict a future state based on previously seen ones. We tested the Markov Model family algorithms: Markov Model, Prediction by Partial Match, LZU and ALZ. We performed full tests of these algorithms with ten users in 3 different types of datasets (communication, application used and location). We then realized that these algorithms limited us to single variables prediction rather than combined variables, i.e. context states.

Later on, while doing a research project at Microsoft's Search Technology Center in Europe, we realized that the type of machine learning algorithms that are more suited to infer context states using context variables were classifiers. At the time I was part of the program management team in the UK center in London, researching product strategy for search technology and its application towards next generation search engine activity. I was researching on augmenting queries with context to help improve the search results, focusing on modeling user's input and domain categories to match queries with algorithmic search results.

It was during my stay that I realized the potential of applying the context-aware model and underlying architecture we were working on at Bing to improve understanding of the user intent when performing a query, and decided to use mobile search engines as examples on which to apply my model. We researched on this topic and defined how to implement our proposed model for a mobile search engine. We published the result of our work in the article "Augmenting Mobile Search Engines to Leverage Context Awareness" (Yndurain,

Bernhardt et al. 2012) , responding to the CFP “Beyond Search: Context-aware computing” on the special issue on context-aware computing of the IEEE Internet Computing of March-April 2012.

It was also during my stay at Bing, after seeing the potential of classifier machine learning algorithms to group data into classes that I understood that I needed to use those algorithms to infer context. I did some research on the best performing classifiers which I contrasted with my peers at Bing and decided to select an array of different types of algorithms. However, prior to being able to test the algorithms I had to create a model that would normalize context data into a unified format transforming the context signals into variables that the classifier would understand. This work was what lead to create a context-awareness unified model as part of the architecture.

1.6 Keywords

Context-awareness, mobile computing, intelligent systems, machine learning, data modeling, pervasive computing and mobile search.

2 State of the art for context-awareness

New ubiquitous computing applications based on intelligent systems are designed to be aware of the context in which they run. The applications require special properties, such as capturing and understanding real-world information and being self-adaptive and proactive (Albrecht Schmidt, Michael Beigl et al. 1999), which traditional computing applications do not support. Ever since the concept of context awareness was identified, it has been the subject of significant academic interest. However, in only the past few years have some large corporations (e.g., Cisco Systems, Nokia, and Samsung) begun to explore the potential of context awareness for context-enriched services (Coutaz and Crowley 2005).

We focus our research on mobile devices and, more concretely, smartphones, because they are able to capture contextual information, such as location, interaction with the mobile device, and surrounding information (i.e., video, audio, noise level, and temperature). Therefore, a primary enabler for the adoption of mobile context awareness is the proliferation of sensors and their inclusion in new smart devices. Sensors capture contextual information from the environment and convert it into signals that can be read and transformed into metadata. Sensors are now routinely included as features of many portable devices. For instance, in 2009, there were 435.9 million accelerometers, gyroscopes and pressure sensors in mobile devices; this number is expected to increase to 2,2 billion in 2014 (Corp. 2010). The processing power of mobile devices has also increased, and broadband connections to mobile "cloud computing" services are becoming more common (Gómez-Barroso, Compañó et al. 2010). ıncluding context information has sparked an explosion of new mobile application development (Thomas 2011), and many mobile devices already include some simple context functionalities, such as functionalities connected with the inclination or movement of the device. Currently, context-aware applications are typically commercialized as mobile augmented reality services in which information from the virtual world (such as the Internet) is superimposed on physical objects and can be browsed with specific software. Such services are a promising industry and are expected to earn revenues of greater than $700 million by 2014 (Juniper 2009).

However, there is still a lack of understanding of how to develop context-aware mobile systems. In this chapter we provide a history of context-awareness computing, and we review some examples of context-aware mobile applications and analyze their underlying architectures and computational models in an attempt to understand the primary challenges in making the concept-aware computing concept mainstream.

In this chapter, we review the definitions of context-aware computing and the existing examples of context-aware mobile applications using several case studies. We also introduce design principles to contextualize mobile applications and analyze the architecture

approaches used in various context-aware systems. Finally, we conclude the chapter by discussing the primary findings of this work and outline some of the future areas of interest for mobile context-awareness.

2.1 Context definitions

Since the 1960s, there has been an on-going debate about the definition of context in relation to computer systems. The term context, or context-awareness, usually refers to a general class of systems that can sense a continuously changing physical environment and provide relevant services to the user based in response to the environment (Dey 2001). However, according to Massimo Benerecetti, "we are very far from a general unifying theory of context" (Massimo Benerecetti 2008) .

The word context as defined in the dictionary is a noun that derives from the Latin "contextus", which means the "connection of words, coherence. From contexere to weave together, from com + texere to weave" (Merriam-Webster 2008).

Two definitions are specified in Webster's on-line dictionary:

- "Definition #1: the parts of a discourse that surround a word or passage and can throw light on its meaning".
- "Definition #2: the interrelated conditions in which something exists or occurs".

Based on this core definition, many authors have proposed their own definitions which we group based on the approach they take: knowledge, cognition and semantic. We study each approach in depth.

2.1.1 Knowledge based context-analysis

In (Penco 2008), Carlo Penco separates context analysis into social and individual sharing. It is believed that there is a set of knowledge shared between a community and not every individual of that community possess all aspects of that knowledge; this scenario is known as social sharing. Another situation occurs when all of the people have the same information or background; this scenario is known as individual sharing. This type of sharing can be subjective (implying awareness), objective (descriptive facts) or normative (information that people should have). It is necessary to distinguish among these different aspects of sharing and to clarify how much of information is shared in which ways.

Another concept that Carlo Penco describes is that of external and internal norms. External norms describe what a speaker should think given his or her beliefs. Internal norms are "merely subjective reflections of the external norms" (Penco 2008). For external norms, the discourse context is provided by shared assumptions; for internal norms, the discourse

context is provided by what the interlocutors assume to be shared assumptions. Both of these norms can be used to understand context because observers cannot know in advance how a situation might change. In these situations, we can either reject any assertion or accept it by using conditions of justification (Penco 2008).

Thomason also uses the above approach when analyzing context as a knowledge source to model it in a type-theoretic setting to identify it with the set of propositions that it delivers. This approach is very useful for an application that is not an agent on itself to simultaneously accesses information from many contexts. However, if we wish to allow context to access information from other contexts, then we require an enriched representation of contexts (Thomason 2008).

2.1.2 Cognition based context-analysis

Philosophers have long discussed the effect of context, Gottlob Frege (Stoke 2003) claimed that "the meaning of a term can only be given in the context of a sentence". There must be some essential connection between what we say and what exists.

In (Hinzen 2008), Wolfram Hinzen claims that to give a definition of context, we must first analyze each aspect of cognition separately to understand what the notion and the role of context is. He focuses on "decontextualizing", which is the process of "interpreting an utterance in context". To do this, he uses logic to analyze truth-conditional interpretations of assertions to assign values to elements in the expressions uttered. He claims that "in a context system of linguistic knowledge interfaces with human intentions, background assumptions, culture and custom, a perceptual situation, the climate, human temper, and more." He argues that "efficient communication does not seem to depend on an exact matching of interpretations."

Wolfram Hinzen claims that a crucial fact of human cognition is that speakers present notions in specific contexts. He believes that context variables have to be employed in a fully general manner because expressions are usually associated with a context (Hinzen 2008).

2.1.3 Semantic based context analysis

Semantics, the formal study of meaning, has also attempted to explain linguistic meaning in terms of the relation of a sentence to the mind of a speaker who utters it (Hinzen 2008). Hinzen claims that a formal representation of a discourse can be interpreted as content (a set of truth conditions) as well as context (for the interpretation of further input). A sentence's meaning itself consists in a dynamic potential for changing a context. He writes, "Understanding is a creative and deliberate process, not a casual one". The speaker's intentions can be recognizing only if they cooperate (Hinzen 2008). Hinzen considers context

related to meaning and argues that meaning is a change of context that is related to propositional attitudes (such as beliefs and presuppositions). When determining beliefs, we make assumptions concerning which beliefs would be rational for a person to have in the given context; that question guides us in determining which belief she or he actually has.

This concept leads Hinzen to analyze propositions to make distinctions between them. The types of propositions are:

- Beliefs: These are propositions that are judged as fully true and that form a background for reasoning and inquiry that is not questioned.
- Potential beliefs: These are propositions that are, relative to the background propositions, judged uncertain or probable, where the probability is subjective.
- Outcomes: These are propositions that are judged valuable using a system of value commitments.

Related to his discourse on context change, Hinzen writes that "if a context changes throughout the assertion of a sentence, it does so in two ways: first the set of mutually shared assumptions adjusts to the fact that a particular sentence with a particular content has been asserted and secondly the proposition proposed for acceptance is either accepted or rejected."

Context variables must be employed in a fully general manner because ranges of quantification of general expressions are usually restricted to a context (Hinzen 2008).

Similarly, Ora Lassila et al. define a representation and reasoning framework for context awareness that employs descriptive logic and associated interference to model and process context data (Lassila and Khushraj 2005).

2.2 Context awareness overview

Context awareness originated as a term from ubiquitous, or pervasive, computing that sought to address linking changes in the environment with computer systems, which are otherwise static (Dey and Abowd 1999). Context-aware applications have existed since the early 1990s, when they were created at Xerox PARC as part of the pervasive computing project (Brown, Bovey et al. 1997). The initial motivation for context-aware computing was to reduce the explicit information a user requires when interacting with a computer (Nicholas D. Lane, Emiliano Miluzzo et al. 2010). In general, context-awareness has gained importance in software development, and it is currently used to enhance applications, although still in modest ways. The usual procedure involves adding meaning to the user's input by including real-world information that can help the system understand explicit input and make interactions more personal and effective. In fact, we engage in this process of contextualizing continuously and unconsciously when interacting with humans or

computers. Transforming the process into a conscious process with the help of technology can make interactions more effective (Gellersen, Schmidt et al. 2002).

2.2.1 Context-aware computing

Context-aware computing refers to a general class of mobile systems that can sense their physical environment (including space and place), i.e., their context of use, and adapt their behavior accordingly to take advantage of contextual information (Chen and Kotz 2000). Context-aware applications are those that change their behavior according to the user´s context, and they are mainly driven by the user's context (Brown, Bovey et al. 1997). In general, context-aware applications are primarily focused on triggering and passing the information to a program that is able to process it.

Context can provide a way to improve human-computer interaction by giving additional meaning to the user's input, which combines explicit and implicit human-computer interaction. According to Albrecht Schmidt et al., three application domains in which context can be beneficial can be distinguished: adaptive user interfaces to make interaction easier, context-aware communication that manages calls, and proactive application scheduling that adapts to the user's situation (Schmidt 2000).

Context awareness can be applied to applications in two ways:

- Orientation to what is being performed (retrieving information or executing a command).
- The means of task execution (manually or automatically).

The possibilities to introduce creative and useful mobile applications or web services that build upon real-world data are numerous. These services can range from usability improvements to industrial applications. Even very simple environmental variables can lead to novel solutions when integrated into mobile devices.

Context-aware applications look at the: who's, where's, when's and what's the user doing of entities and use this information to determine why the situation is happening. It is not the application that determines why a situation is occurring; instead, the designer of the application must do this and use context to make decisions accordingly (Dey and Abowd 1999).

Albrecht Schmidt and Kirstof Van Laerhoven (Schmidt and Laerhoven 2001) suggest terminology to consider when analyzing context awareness:

- Situation or situational context that describes the real-world situation.
- Sensor data or situational data that describe the data captured to represent the situation.

- Context or context knowledge to describe the abstract description of the real-world situation.
- Context-aware applications or context-sensitive applications, which refers to items that change their behavior according to context.

Context in mobile computing has two different aspects. One aspect includes characteristics of the surrounding environment that determine the behavior of the mobile applications. The other aspect is relevant to the application but not critical because it is only necessary to adapt 'the display of information' to interested users. Schmidt and Laerhoven define these aspects as active and passive.

A system is context-aware if it uses context to provide relevant information or services to the user, where the relevancy depends on the user's task (Dey and Abowd 1999). Context information is derived from an array of diverse information sources, such as sensors, computations, networks or human interaction. There are two categories of parameters that intersect to create contexts, action and surrounding-related.

There are some mechanisms to sense context and deliver it to applications. Guanglin Chen and David Kotz define the following mechanisms in (Chen and Kotz 2000):

- Sensing location, which can be either indoors or outdoors. The problem with this method is that there is no uniform method to track locations with fine granularity that works either indoors or outdoors, especially when there is low coverage.
- Low-level contexts, which include time, nearby objects, network bandwidth, and orientation, among other properties.
- High-level contexts, which define the user's current activity using rule-based systems. Chen and Kotz find that this system is ambiguous because it lacks boundary conditions and is an undefined model.
- Sensing context changes that are based on a publishing analysis method that they believe can be used for monitoring context as well.

The raw data may be sufficient for some applications, but other applications require the data to be transformed or fused with other data to derive higher-level context for more accuracy. Typically, context has to be provided explicitly, although it can also be derived. We define three common phases in context-awareness computing (Figure 1) that we have observed while analyzing and reviewing the current academic literature:

1. Context sensing: All relevant information about the user's environment is sensed. An easy example to understand this is location sensing via a GPS.

   In this phase, it is important to define what information should be sensed and what the available sensors are.

2. Context refinement: Analysis of the information by combining it (or aggregating it) to generate concrete context information from attributes that are offered by available context sources. This includes understanding the data and relating those data to their context.

   There are some refinement techniques that can be used, such as combination, deduction, filtering, and interpolation or extrapolation.

3. Context dissemination: Usage of context once it is refined to adapt to the content it provides. Rules for action triggering are defined in this phase. Sometimes there is an extra step in this phase, which is the evaluation of the success rate of the context-aware application.

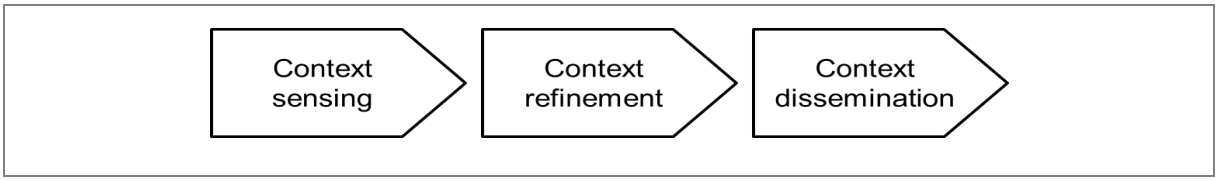

Figure 1 Context-aware computing phases

Current research focuses on one or more of the phases to propose context-aware solutions. Taxonomy about context awareness is defined by grouping the actions that happen in each step as shown in Table 1.

	Context sensing			Context refinement		Context dissemination	
Action	Obtain data from sources	Measure data quality	Model data usage in contexts	Understand and relate data to context	Integrate application into system	Rules to trigger actions	Execute context-aware actions

Table 1 Taxonomy of context awareness steps

When implementing context aware systems, the idea is to use sensor data, process it and predict the context, as shown in Figure 2.

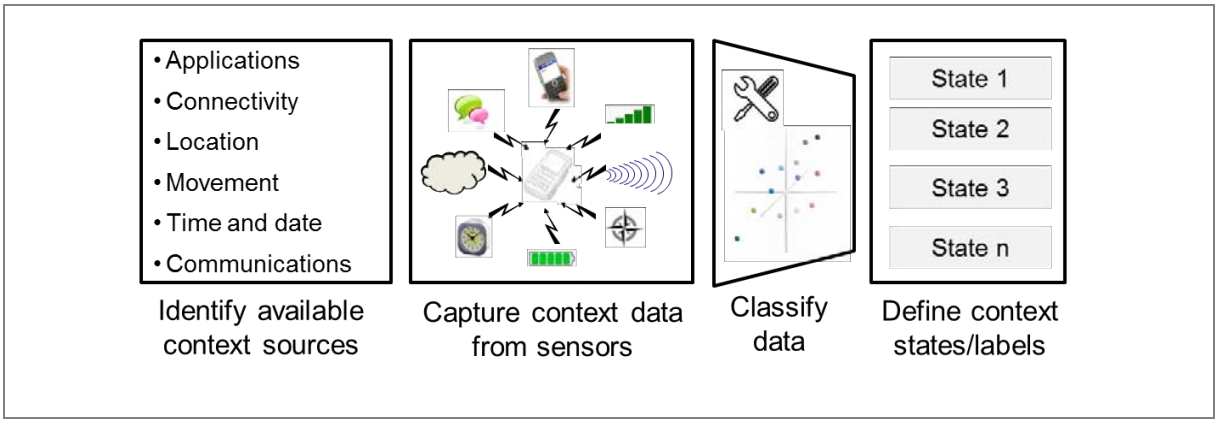

Figure 2 Conceptual diagram of how to capture data to define contexts

Hans Gellersen et al. (Gellersen, Schmidt et al. 2002) consider that context is derived by obtaining data from the physical surroundings of a device, capturing the data, and analyzing it. Devices have direct or indirect awareness of context. The authors argue that a smart device is one that is not ignorant about its environment; this notion results in a wider notion of context than that directly related to the usage of the word in common language. Most of the adaptive applications require similar contextual information, which are either the basics (such as location context) or high-level social context (such as user activity).

2.2.2 System components

Context is formed from several components related to mobile devices. Some components can be the device's internal components, some can be interactions with the user or with other devices, and others can be related to the environment. Consequently, the following statements are true:

1. There are many ways to classify context data that depend on which perspective we take. For example, we can think of context from an end-user point of view, and therefore we would consider what the user is doing with the mobile device. Or, we could think of context from a mobile device point of view; we would then analyze what is happening to the device itself.
2. There exist different information sources from which we can obtain context-related data. Data can be derived in a straightforward manner, from one single source, or by combining several sources. We refer to this as simple or complex context. Context data sources can be mobile system components, e.g., the connection between the mobile device and the network, or an action, e.g., texting and making phone calls.

Context-aware systems are composed of a combination of hardware and software components, which can be either internal or external to the device where the application runs; we define such a system in Figure 3.

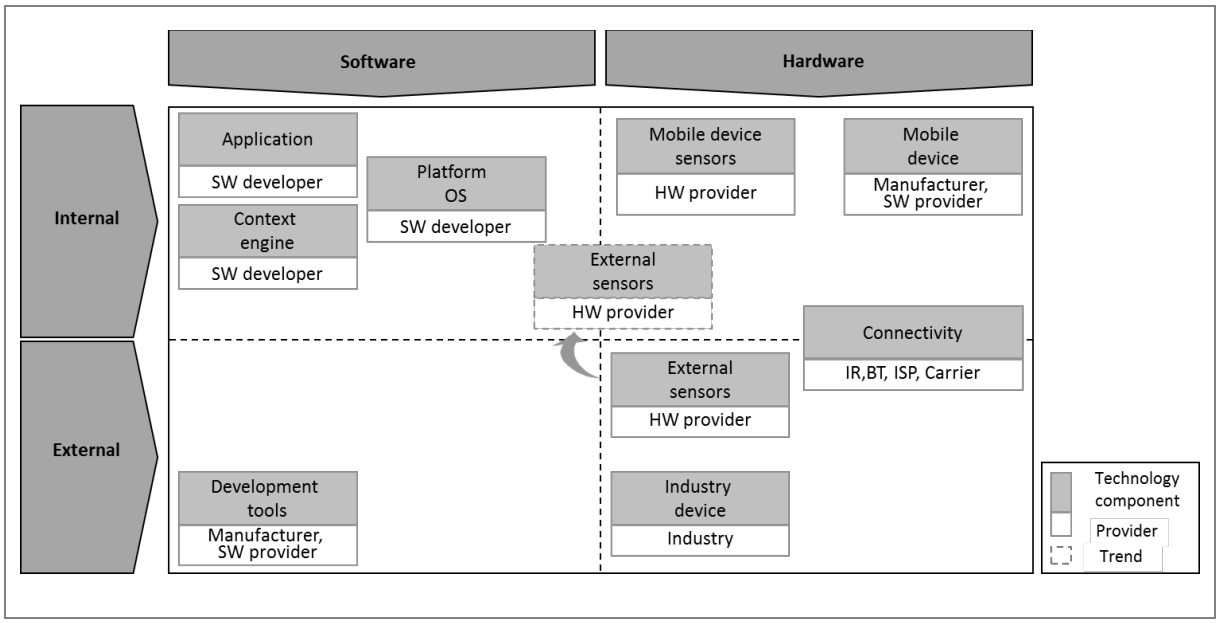

Figure 3 Context-aware ecosystem matrix

Each technology component is typically provided by a different manufacturer.

- Software applications that are context-aware enhanced are often provided by the developer.
- Context engines that recognize context and make use of it in applications, based on artificial intelligence techniques or reasoning, are also provided by a (usually different) software developer.
- The mobile platform operating system (OS) where the application runs can either be an operating system or a web browser, which are provided by software or computer enterprises.
- Sensors that capture contextual information can be inside or outside the mobile device where the application is running, although the trend is to embed as many of them as possible until the Internet of Things concept has been better developed and implemented. The sensors are provided by diverse hardware manufacturers.
- The mobile device where the application runs (PDA, netbook, smartphone) is provided by a handset or computer manufacturer.
- Connectivity allows connections to other consumer electronic devices that can be located as close as 10 cm or thousands of kilometers away by means of different types of connections, such as RFID, Bluetooth, infrared, WLAN, UMTS, LTE, WiMAX, among others, which are provided by carriers, ISPs or sensor manufacturers (such is the case of RFID).
- Industry devices retain contextual information provided by consumer electronics manufacturers.

- Development tools are provided through a software development kit (SDK) to help developers; some additional tools are provided to help program applications.

2.2.3 Inherent challenges

The obvious highly heterogeneous nature of context-aware application components poses several challenges to developers:

Context data quality and heterogeneity

The problem with sensed contextual information is that it is highly dynamic (and thus prone to noise and sensing errors), and while user-supplied information is reliable, on many occasions, it is out of date or not useful. This situation results in a lack of data quality that limits the performance of context-aware applications. The underlying problem is that two pieces of context information can differ in their precision, correctness and trustworthiness (Buchholz, Kupper et al. 2003). Therefore, there is a need to perform data completeness, verification and correction checks (Bernardos, Tarrío et al. 2008). These checks increase the complexity of application design and the computational burden.

Additionally, the heterogeneity of contextual information also increases the complexity of the data analysis required to develop applications. Moreover, there is no unique solution for how to blend time and events to understand context changes. In summary, a logical model that includes all existing components of a context-aware mobile computing environment does not exist. Therefore, there is a need to define a specific, context-aggregation mechanism (Lapkin 2009) that includes data classification based on the nature of the context and most likely also to define a formal context-aware development language or interface development environment beyond the APIs (application programming interface) available in a given platform's software development tool kits.

Platforms' technical limitations and contextual processing needs

Mobile devices now merge information technology, telecommunications and consumer electronics for new uses, which transforms them into handheld computers. This increased mobile functionality demands an increasing amount of memory, faster performance and continuous wireless connectivity, which in turn requires higher energy consumption and processing power. This requirement is one of the major challenges for any mobile user technology and is not specific to context awareness. However, context-aware applications will require high processing power and higher capacity batteries to extract the relevant features of context, process context continuously, and maintain permanent connectivity with local wireless networks and with the mobile cloud.

Contextual applications require on-the-spot reasoning and high processing power, but current mobile devices are not suitable for this. Mobile hardware has limited battery life, memory and processing speed, all of which are features very much needed to capture

information from sensors, extract the relevant features of context, process the context continuously, and maintain a relevant historical dataset that will allow machine learning algorithms to predict context.

Sensor data interoperability and standards and device platform fragmentation

The number of sensors used for context awareness increases daily. The sensors use myriad proprietary schemes, standards and protocols, and there are no widely adopted common solutions for data, formats and communication with the sensors. Therefore, there is a need to develop an ad hoc method to access each sensor's data, which also changes depending on the mobile platform manufacturer.

Furthermore, there is also a wide variety of mobile platform OS, each with a different strategy for application development, which imposes a high transaction cost to develop an application. There is no horizontal SDK framework compatible with all of them. Moreover, in some cases, there is not even compatibility within the same platform, which requires new development altogether for different versions of the same OS and for different hardware supporting the same platform.

Privacy issues

Privacy, along with security and data protection, is one of the major concerns for context-aware applications such as location-based services (Feijoo, Pascu et al. 2009). In fact, one of the challenges in adapting applications to context is that even if the application is reliable, it can be considered by users as intrusive (Korpipää and Mäntyjärvi 2003). Tools and methodologies are needed to define a design of suitable privacy policies for application development to protect users from possible abuses. Several strategies are being discussed; they include: privacy by design (i.e., requiring intervention and explicit acceptance from the user), privacy by law (i.e., defining the sphere of personal data not subject to use by applications) and user empowerment (i.e., keeping users in complete control of their personal data by enabling them to switch on and off the application and data access). One way to avoid privacy issues is to define an architecture in which the context capturing and processing is performed on the mobile device rather than on external platforms. This measure ensures that the user's data are not shared with external parties.

2.3 Context-aware applications

The possibilities to introduce creative and useful mobile applications and web services that build upon real-world data are numerous. Context-aware computing has significant potential uses in a number of areas, and context will be the defining principle of mobile business for the next decade (Thomas 2011). Context-aware applications can range from usability improvements to industrial applications. Even very simple environmental variables can lead to novel solutions when integrated into the mobile device. An application may be interested

in a single part of, many parts of, or the entire context. The goal is to obtain features that describe a concept from the real world so that the context-aware application can utilize it.

We review some examples of solutions running on different mobile devices (PDAs, netbooks or phones) applied to different industries and domains: transportation, health care, automotive, academics, gaming, search engines, social networking, and general usability improvements. Some are pilots in the research phase, and others are commercial solutions used in industry.

When analyzing case studies, we focus on the following elements: motivation, frequency and involvement. The motivation of making the application context-aware refers to how contextual information is utilized. The frequency of context-awareness is how often contextual information is retrieved and applied and whether it is used based on events or periodically, i.e., in a discrete or continuous mode. The involvement of the user in context awareness is considered to be active or passive (Barkhuus and Dey 2003): the user can either request an action based on the context, which is called a pull mode, the context-awareness adaptation can be performed actively without the user's approval (a push mode), or there can be an intermediate push-pull mode in which the action is initiated independently but requires user confirmation.

2.3.1 Case study 1: improving interface usability

Data from mobile phones' or other devices' user interactions and environment are used as input to improve the devices' interfaces and usability by shifting among pre-defined profiles with the goal of reducing the need for the user's attention and targeted to adapt to the phone's environment (context) to improve usability.

In all cases, context-awareness is performed in a push mode in which the device or the application adapts itself automatically without waiting for confirmation from the user. Context data are captured through the mobile device's internal sensors, which include the microphone, the camera, tilt, touch and IR. Contextual information is provided constantly during the use of the application, and the application automatically senses contextual information.

Examples of pilot applications are automobile infotainment applications (Sharma, Kuvedu-Libla et al. 2008), context-aware phones (Siewiorek, Smailagic et al. 2003), (Schmidt and Laerhoven 2001), and adaptive note pad application (Schmidt 2000). We provide an overview of each of them.

Infotainment applications

The application considers the driver, vehicle and environment and adapts the car navigation system to the driving context. It adapts its navigational information in a user-friendly manner

by adjusting the font size according to the speed of the vehicle or increasing zoom levels based on the location.

Context-aware phones

One of the applications is based on an external module, SenSay, attached to a mobile phone and a microphone so that it can compile sensor information and decide which state the user is in (active, idle, normal or interruptible) so that it can adapt the phone ringer and hence reduce its intrusiveness (Siewiorek, Smailagic et al. 2003).

Another of the applications adds context-awareness to a phone's profile option to adapt the profile to the phone's position: if the user is holding the phone, vibrating mode is activated. If the phone is lying on a table, its volume is decreased. The phone's volume is increased if the phone is on a pocket or outside (Schmidt and Laerhoven 2001).

Adaptive note pad

This application proposes to adapt the note pad application of a PDA to its context by changing the font size, backlight, and on/off settings. It adapts the backlight to light change in the environment, it changes the font size according to the movement level of the device, and it turns on the notepad automatically if the user holds the device.

2.3.2 Case study 2: improving industry solutions

Data from a device's location and user interactions, possibly including biometrics, are used as inputs to solutions in specific domains such as automotive, advertising, academics, games or search engines. In all cases, the user must have some active role, which includes at least launching the application to trigger the context awareness. This activation is typically based on environmental and location information. Being context-aware improves the solution by providing specific contextual data that are often key elements in the solution's functionality. Context data are captured through different communications systems and connectivity sensors, such as location systems (GPS), short-range communications (such as Bluetooth), mobile communications (such as GPRS and UMTS) and WLAN. Context awareness is maintained over time, and during utilization, the application is continuously looking for contextual information. Typically, context plays an active role in a push-pull mode in which prior confirmation from the user is requested.

Examples of pilot applications are a parking assistant for an automobile (Grauballe, Perrucci et al. 2008), a mobile advertising system (Aalto, Göthlin et al. 2004) and an education solution (Anumba and Aziz 2006). Example of commercial solutions include mobile game applications such as the game Alien Revolt by M1nd Corporation launched by the carrier Oi (Silva 2008). We provide an overview of each of them.

A parking assistant

This application is a parking sensor application that integrates a mobile device in a car connected to ultra-sonic distance sensors wirelessly. The mobile phone is placed within the car and shows the distances to the nearest objects thus aiding the driver to park properly.

Mobile advertising

This application is a location-aware mobile advertising application, called B-MAD (Bluetooth Mobile Advertising) that delivers location-aware mobile advertisements to mobile phones. Users subscribe to the mobile advertisement service. The application uses Bluetooth positioning to locate the users and the Wireless Application Protocol (WAP) to push the advertisement to them.

An education solution

This application is a context-aware system that runs on a mobile phone that supports students and lecturers at Loughborough University through several functionalities. It delivers the relevant learning content for each student, giving them the appropriate access to on-line resources based on their contextual information, and allows students and lecturers to interact during tutorials.

The mobile game Alien Revolt

This application links a cell phone's connection to physical spaces and users to create an urban, location-based, and hybrid-reality mobile game. The game takes place in the cell phone's screen and the city space, and players can detect other players within a specific radius in the city space to shoot them.

2.3.3 Case study 3: improving workplace spaces

Data about location and user interactions are used as inputs to improve workspaces in domains such as construction sites, train stations, hospitals and offices. Context-aware applications store information about the workspace and display it to the user on demand based on the user's location. Being context-aware facilitates the task management while minimizing the interaction required of the user. Context data are captured through connectivity sensors such as IR, WLAN, or RFID. Context awareness is maintained over time; during the entire (working) day, the application receives contextual information and updates its data accordingly. Context plays an active role in the push mode, where the system automatically incorporates contextual information without asking for validation from the users.

Examples of pilot applications are workforce management at a construction site and workforce management and an information system at a train station (Anumba and Aziz 2006), health care context-aware computing in hospital work (Bardram 2004) and general adaptive workspace applications (Schilit, Adams et al. 1994). We provide an overview of each of them.

Workforce management

This is an application that facilitates work tasks for construction workers by using their context information. When they arrive at the construction site, the network detects their IP, logs them into the system using their mobile phone and pushes their task schedule for the day. The system detects when deliveries are performed using WLAN tags and updates itself periodically.

Another similar application used in the same environment personalizes the interface based on who access it and pushes real-time train information and security alerts to the handheld devices. Based on the devices' location, the application shows them the closest IP-based surveillance cameras and warns the most appropriate (closest) person or object.

Health care

This application is a context-aware application that links electronic patient records, pill containers and hospital beds together using RFID tags and LEDs as sensors. The application can run on different mobile devices (netbooks, tablets or PDAs). The hospital bed and pill container are equipped with computing capacity and sensors. The applications reveals themselves to the person who is in the room (the doctor, patient or nurse) and display the pertinent information (the pill to take or the patient's record).

Workspace adaptation

This is an application that runs on the ParcTab small hand held device that uses IR, the cellular network, and badges placed throughout the environment on walls and objects to sense information. The application detects the objects that are of interest to the user and determines the existing components so that it can adapt to them. Depending on the context information, the application adapts the information that it displays and the actions it takes.

2.3.4 Case study 4: improving search engines

Data from the location, the environment, user interactions and bio-parameters are used as input to improve the results of search in mobile situations. Context data are captured by sensors embedded in the mobile device that extract information from the surrounding environment and that are able to communicate with other sensors (aka the Internet of

Things). The users' profiles and their behavioral patterns are used to make the search yield more meaningful results. Context awareness is activated when a query is performed in the search engine application, and depending on the application, further intervention from the user (pull mode) can be required; intervention is particularly necessary if privacy or personal data must be compromised.

Examples of proposed and pilot applications include improving the interaction paradigm through new ways of processing queries and categorizing results, as in Yahoo´s oneSearch; adapting the user interface display, as in Yahoo´s oneSearch (Yi, Maghoul et al. 2008) and the Findex browser (Heimonen and Käki 2007); contextualizing queries (Church and Smith 2008) and the AroundMe search engine that uses local search (Perez 2012) (Inc 2011). These proposed applications all run on mobile devices and smartphones. We provide an overview of each of them.

Yahoo's oneSearch

oneSearch is a mobile search service with three interfaces built on top of it, a browser, an application and text messages. It improves the user experience by providing immediate answers to user queries, by minimizing the number of clicks needed to search, and by using context data to categorize queries.

Mobile Findex

This is a mobile search engine interface that uses Google's Web API and that uses context to rank the results of a search to make them more relevant and to adapt them to the mobile screen size. The user interface has three views, query, category and result, available in the heading.

Contextualize queries

These are mobile search application whose interface includes contextual information, such as location and social information, to adapt the search experience. The search engine shows the recent queries and search results submitted on different locations marked on a map, allowing users to filter queries based on contextual data.

AroundMe

This is a mobile search application that uses location context data to find information about the surroundings of the user. It then matches the information with the query to output the closest place. It outputs a list of nearby locations and allows the user to share the list or find further information about those places using Wikipedia.

2.3.5 Case study 5: improving social networks

Location data are used as input to improve the experience of mobile social networks. Social network applications allow users to share their whereabouts with the contacts in their network by adding their location to their status or their news feeds. This feature improves the social networking application by adding location information in real time. Users check into places and can share where they are and make comments about the place. Context data are captured by the device's embedded GPS or by the user's input through the keyboard. Context awareness is activated when the location sharing application is launched, and it requires intervention from the user to comply with privacy rules.

Examples of existing commercial applications are Facebook Places[i], Google's Foursquare [ii] and Gowalla [iii] . All of these applications enhance social networking by adding location information that informs users of where their friends are at any given moment (Sharon 2010). These tools run on most mobile platforms, including RIM, Android, iPhone, Palm, and others.

2.3.6 Case study 6: improving marketing campaigns

Contextual information is used to place an advertisement on a page and to launch a specific campaign aimed at the user. Multimedia information and relevance are used as triggers to contextualize advertisements based on user's profiles. The application pushes contextualized campaigns and advertisements based on the user's profile and the type of device on which the application runs. This feature improves the advertisement by making them more relevant to the user while reducing intrusiveness. Context data are captured by the mobile browser using the device's embedded connectivity channels (Bluetooth, WLAN, and GPRS). Context awareness is activated using semantics in the advertising engine that are used to match the advertisements to the user profile. Examples of existing commercial and pilot applications are the System for Mobile Marketing (Kurkovsky, Zanev et al. 2005), Media Sense (Mei and Hua 2010) and contextual semantic advertisement (Broder, Fontoura et al. 2007). All of these applications enhance ads by adding contextual information to personalize the ads according to the user's profile. These tools run on most mobile platforms, including PDAs and any mobile browsers running on mobile devices. We provide an overview of each of them.

SMMART

This is an application that sends targeted promotions to mobile device users based on context information. The application adapts to the changing interests of its user by monitoring shopping habits. The prototype built runs on a Pocket PC.

Media Sense

This application provides contextual multimedia (image, video, and game) advertising in which the ads that are placed on a page are relevant to the available media content and surrounding text rather than the entire web page. The advertisements are dynamically embedded at the appropriate positions within each multimedia object rather than pre-defined and static.

Contextual ads

A contextual ad application places commercial ads within the content of a generic web page and aligns the ad with the content of the page with the goal of providing a better user experience and increasing the probability of clicks. The system processes the content of the page, extracts features, and then searches the ad space to find the best matching ads. The application runs on any browser.

2.4 Context-aware models and prediction

In general, authors include in their architecture proposals two main common functions: one is a context model that, as a knowledge base, is populated with facts, rules that associate facts and pre-defined situations, and sensors capable to trigger events that are used to update the assertions and reprocess the reasoning according to the rules. The other function is a context prediction technique that, using the contextual data inputs, is able to derive context states. We describe some examples from the current literature that focus on these two topics.

2.4.1 Modeling context

As context-aware computing advances, there is a need to describe a formal context model that focuses on transforming contextual data into a "plain mobile natural language." So that the data can be understood and used properly by the application, an ontology with the appropriate vocabulary is required to act as a semantic framework that translates context data into context information (Korpipää and Mäntyjärvi 2003; Gu, Wang et al. 2004; Wang, Gu et al. 2004). To do this, we use ontologies because they are ways to specify a conceptualization with the goal of sharing "knowledge and interoperation among programs based on a shared conceptualization" (Gruber 1992), and they will allow us to model context information.

Ontologies provide a way of categorizing information and giving meaning to it, whereas vocabularies store the definitions; together, they form a semantic framework that provides

the meaning of the data (Henricksen, Indulska et al. 2002; Korpipää and Mäntyjärvi 2003; Wang, Gu et al. 2004) and are generally based on the following principles:

1. The information domain is restricted to the range of meanings emanating from context data sources.
2. The ontology design should be simple so that anyone can understand it.
3. The information should be accessible so that the queries are as straightforward as possible.
4. The design should be as generic as possible to support different types of context data and to support different types of inference for queries.
5. The design should be flexible so that it can cater to all possible user interests and situations as required.

Each context variable is defined using a group of values (mandatorily or optionally defined) with some properties that form a context category as a verbal description (see the examples in Table 2).

	Property	Description
Mandatory	Context type	Category of context
	Context type level1	First level of context that gives specific meaning to the context category.
	Context type level2	Second level of context that gives specific meaning to the context category type.
	Value	Raw numerical value that context variable takes (type + subtype).
	Timestamp	Latest time the context value was retrieved.
Optional	Probability	Degree of quality/confidence the context value has.
	Source	Where does the information come from, where is it located.
	Attributes	Any other information that might be useful.

Table 2 Common context ontology variables

The context model is structured around a set of abstract entities that describe a physical or conceptual object and the relation within entities. It uses logic to derive high-level and low-level contexts signals. Explicit context is acquired, whereas implicit context is deduced from explicit context (Wang, Gu et al. 2004).

Context models define the dependencies and relationships between entities and attributes (Henricksen, Indulska et al. 2002) and are able to capture all characteristics of context information to be used as the basis to determine context (Gu, Wang et al. 2004).

2.4.2 Context predictors

We also review some existing context prediction approaches available in the academic literature that are similar to the goal of our research, which is to predict context in a mobile environment.

MavHome

The MavHome smart home project is a research project at the University of Texas at Arlington that has the goal of creating an intelligent and versatile home environment. The house must be able to adapt to its inhabitants based on an understanding of their activities. The smart home inhabitant typically interacts with various devices as part of his routine activities. Such interactions may be considered as a sequence of events with some inherent pattern of recurrence; an example is the breakfast routine. In the project, the MavHome coverage area is partitioned into zones or sectors, and the smart home must locate an inhabitant in the home area within the MavHome environment and predict the inhabitant's next action so that the smart home can automate the routines and repetitive tasks to the inhabitant (Das, Cook et al. 2002). The project uses several on-line prediction algorithms based on Markov models to learn patterns and to determine the future location.

Evaluating Next-Cell Predictors with Extensive Wi-Fi Mobility Data

Song et al. test several on-line prediction algorithms to learn patterns and to determine the future location (Song, Kotz et al. 2006) based on empirical evaluation of location predictors using a two-year trace of mobility patterns of users of the Dartmouth campus's Wi-Fi wireless network.

In the project, the researchers compare the prediction accuracy of three predictor algorithms, Markov-based, compression-based, and prediction by partial match and sampled pattern matching.

The researchers assume that users reside at a given discrete location at any time, and they partition the college into areas based on the wireless cells that cover them and use the movement history as the input for the algorithm to predict the next location. The history is a sequence of location observations of the user recorded periodically. The predictors are domain independent and consider the locations as symbols without taking into account other semantics.

The researchers conclude that low-order Markov predictors performed as well or better than the more complex ones and consumed less storage capacity.

The Alignment Approach for Context Prediction

Stephan Sigg, Sandra Haseloff, and Klaus David study the prediction accuracy of algorithms for context prediction in ubiquitous computing environments (Sigg, Haseloff et al. 2010). They select algorithms that have good prediction accuracy and high prediction horizons suited for ubiquitous environments with low processing and memory requirements and error tolerance for input data. They propose an alignment prediction that relies on typical context patterns, and they abstract from noise input sequences by comparing the most similar ones computed between 2 patterns. They follow three steps: 1. computation of typical context patterns, 2. computation of semi-global alignments, and 3. prediction of the most similar typical subsequence. They test four prediction algorithms (Markov processes, prediction by independent components, prediction by principal components and autoregressive moving average prediction models) using different datasets, wind power prediction and location prediction on raw and clustered GPS data and on the Reality Mining dataset. They analyze the performance of each algorithm for different circumstances (time and number of contexts) and conclude that Markov prediction excelled for short prediction horizons with a high number of nominal contexts.

Human Activity Recognition and Pattern Discovery

Enju Kim, Sumi Helal and Diane Cook analyze several probability models to recognize activities in real life settings. (Kim, Helal et al. 2010). In their research, they propose two approaches, activity pattern discovery and activity recognition, and they test four algorithms (Markov models, the conditional random field, the skip chain random field and emerging patterns) because these algorithms are among the most popular modeling techniques.

In their pattern discovery analysis, they focus on finding unknown patterns directly from low-level sensor data without any pre-defined models or assumptions, i.e., in an unsupervised manner. They build a hierarchical activity model that splits between high- and low-level activities and define the relation between them and the data-extracting rules.

In the activity recognition analysis, they focus on accurate detection of human activities based on a pre-defined model. They build a high-level conceptual model first and then implement it, because they claim that once activities are discovered, they can be used as the basis for a model to recognize the activity and track its occurrence.

2.5 Architecture approaches

There are several architecture and model designs proposed by researchers and practitioners over the last decade to enable context-aware mobile systems and in this section, we review the main approaches and analyze them.

2.5.1 Overview

We describe ten proposals of context-aware mobile systems defined over the last decade's literature. Some are exclusively designed for mobile devices, and others are designed for fixed environments as well.

Ad hoc middleware (Sørensen, Wu et al. 2004)

Carl-Fredrik Sørensen et al. propose a middleware design approach based on a sentient object that captures and disseminates contextual information (Sørensen, Wu et al. 2004). This middleware interfaces between the sensors and the applications that use contextual information. Its goal is to provide a framework that explains how to capture data from sensors and disseminate those data to applications. The middleware consists of the following components:

- The publish-subscribe component is used to discover mobile entities in the proximity.
- The group communication component is used to provide a communication protocol suite.
- The context CF (capture and fusion) component provides the facility for sensor fusion and the inference engine.
- The quality of services component manages the allocation of resources.

Context Modeling Language (Henricksen, Indulska et al. 2002; Henricksen and Indulska 2004)

Karen Henricksen and Jadwiga Indulska present the Context Modeling Language (Henricksen, Indulska et al. 2002; Henricksen and Indulska 2004) as a tool to assist in exploring and specifying the context requirements of context-aware applications. The software infrastructure they provide to support their model is divided into layers:

- The context gathering layers acquire information from sensors and process it.
- The context reception layer maps context gathering and management layers and translates input into fact-based representations.
- The context management layer maintains a set of context models using relational representation.
- The query layer provides applications and the adaptation layer with a convenient interface with which to query the context management system using the fact and situation abstractions.

- The adaptation layer manages common repositories of situation, preference and trigger definitions. It evaluates these on behalf of applications using the services of the query layer.

Solar system (Chen and Kotz 2002)

In their research, Guanling Chen and David Kotz (Chen and Kotz 2002) propose Solar, a system to make use of data for context-aware applications. In their architecture, events flow through a graph in which they become customized context for individual applications. They define how to setup a context-aware system infrastructure by decoupling the application and the context sensing. They propose introducing a middleware to separate the low-level sensor data from the high-level applications. The objective of this middleware module is to collect raw sensor information and translate it to a format understood by the application and disseminate it to interested applications. Two approaches can be taken to define this architecture: (1) a centralized architecture that uses a server that provides contextual information to the applications and (2) a distributed architecture based on a context trigger concept that is installed on the devices.

Context home (Meyer and Rakotonirainry 2003)

Sven Meyer and Andry Rakotonirainry propose a middleware and framework (Meyer and Rakotonirainry 2003) for context-aware applications. It is based on three main components:

1. A hardware abstraction layer that decouples the software from the sensors and communication infrastructure.
2. A context manager to derive basic context information from the raw sensor data maps those data to a suitable context model and derives higher-level context information from lower-level information.
3. A privacy manager that ensures that the only selected information leaves the privacy domain and otherwise ensures that as little as possible gets escapes.

Most functions of these components will be performed by the middleware or some type of framework to make them reusable for all types of context-aware applications. The task of the hardware abstraction layer is to decouple the higher-level software from the actual sensor hardware and to communicate with the network. The authors define a context manager that should be able to combine lower-level context information with higher-level constructs not conceived of at design time.

Semantic architecture (Lassila and Khushraj 2005)

Ora Lassila and Deepali Khushraj propose middleware architecture (Lassila and Khushraj 2005) for enabling context-sensitive behavior and processing in which they model context

using description logic. They present a representation and reasoning framework for context awareness that employs description logic and an associated interference to model and process context data. They contextualize applications via semantic middleware. They define a general architecture as an application middleware. They propose the following information flow:

- Data are collected from sensors and integrated with domain knowledge. The sensors are referred to as data sources.
- The data are used as input to determine the current context; this step is termed context derivation.
- The context information is used to affect context-sensitive behavior (using scripts and production rules); this step is termed behavior integration.

This framework enables the creation of a distributed context-architecture and enables applications and context reasoners to use context.

Context sensing (Schmidt 2000) (Schmidt and Laerhoven 2001)

Albrecht Schmidt (Schmidt 2000) and Kristof Van Laerhoven present an architecture and the necessary mechanisms (Schmidt and Laerhoven 2001) to support the transformation from sensor data to cues to contexts as a foundation to make context-aware applications. They define a layer-based architecture in which sensor data are captured in one layer, another layer transforms the data into cues, and another layer transforms the cues into context so that applications can use the context. The layers are described below:

- Sensors: There are both physical and logical sensors (e.g., light, audio, accelerometers, and location).
- Cues: The cues provide an abstraction of physical and logical sensors, and they are dependent on a single sensor, but multiple cues can be calculated using the data from one sensor. The cues reduce the number of data provided by the sensors.
- Contexts: The context is the current situation on an abstract level, which is derived from the available cues.
- The application layer is the final layer.

ContextPhone (Mika Raento, Antti Oulasvirta et al. 2005)

Mika Raento, Antti Oulasvirta, Renaud Petit, and Hannu Toivonen define a software platform (ContextPhone) that provides the features developers need when creating context-aware applications for mobile platforms (Mika Raento, Antti Oulasvirta et al. 2005). It consists of four interconnected modules based on open-source C++ libraries and source code components:

1. Sensors to acquire context data from different sources.
2. Communications to connect external services via Internet protocols.
3. Customizable applications to augment or replace built-in applications.
4. Systems services to launch background services and provide background information.

The goal of the application, which is a development framework, is to provide context as a resource. It interfaces and integrates with existing smartphone applications such as messaging and calling, and it runs in the background.

Hydrogen (Thomas Hofer, Wieland Schwinger et al. 2002)

Thomas Hofer, Wieland Schwinger et al. propose the Hydrogen architecture framework for context-aware mobile systems (Thomas Hofer, Wieland Schwinger et al. 2002).

It is a three-layered architecture that responds to particular requirements of mobile devices. The architecture has the following layers: adaptor, management and application.

The Hydrogen approach considers context as any pertinent information about an application environment and describes it using an object-oriented model. It has a set of pre-defined context classes that can be extended and reused.

Hofer et al. develop a prototype of the architecture using Java over an iPAQ. The prototype accounts for the limited resources of mobile devices and uses a peer-to-peer communication model. The adaptor layer performs both the context sensing and interpretation tasks.

CASS (Patrick Fahy and Clarke 2004)

Patrick Fahy and Siobhan Clarke propose a middleware-based architecture for context awareness in mobility (Patrick Fahy and Clarke 2004). In the project, mobile devices are equipped with various sensors to perceive context variation and send it to the server without local processing. The interpretation module is installed on the server, which is connected wirelessly to the phone. The architecture is modular, which allows one to easily enhance the system by adding new functions.

MobiPads (Chan and Chuang 2003)

Alvin T.S. Chan and Siu-Nam Chuang propose a middleware for mobile platforms (Chan and Chuang 2003). It is composed of two agents, a server at the wired network, and a client at a mobile device attached to the Internet through wireless or cellular networks. The entire MobiPADS system is implemented and deployed in a Java platform, and it uses an event notification model to monitor the status of the specific context and report the event to the subscribed entities. The model provides an interface for applications to directly participate in adaptation of computation in response to the changing context.

MobiPADS provides flexible configuration of resources to optimize the operations of mobile applications, and it is implemented as a collection of active-service entities, known as mobilets. It supports dynamic adaptation at the middleware and application layers and augments mobile applications with context.

User behavior (Mayrhofer, Radi et al. 2003)

Rene Mayrhofer et al. propose an architecture that allows mobile devices to continuously recognize current and anticipate future user context (Mayrhofer, Radi et al. 2003). It consists of the four major parts of context-awareness: feature extraction, classification, labeling and prediction. Mayrhofer et al. interpret context changes as a state trajectory to enable the forecasting of the future development of and to anticipate context. The architecture is targeted for embedded systems (mobile), and it describes in detail how to extract context data, classify those data and predict future classes. They test their experiment with real life data from three weeks of usage that are stored in a network computer. They do not define the classification labels they use and they do not implement the context class prediction.

2.5.2 Evaluation

Most of the approaches adopt an infrastructure-centered view, assuming that the complexity of engineering context-aware applications can be reduced to the infrastructure responsible for gathering, managing and disseminating context information. The emphasis is on designing an architecture that includes knowledge taxonomy and has the highest possible processing efficiency and programming simplicity. The main differences between the different design options are where to place the context-reasoning engine.

There are three type of architecture approaches, which are those based on layers, middleware or a mixed approach:

- Layer-based design: The context-processing functionality is split between layers that perform the three primary tasks of understanding contextual information, processing it and using it in the application. This approach increases the complexity of maintenance because integration of new context sources must be propagated through all of the layers.
- Middleware design: Contextual information is captured by sensors and directly analyzed by the middleware reasoning engine. In this approach, the processing of context is slower because there is no pre-processing of data.
- Mixed design: The different layers communicate through a middleware context manager. This approach tries to be more flexible than the previous models by facilitating the task of integrating different context sources.

We will also consider if the architecture is specifically oriented to a mobile device.

There are also three different approaches regarding where the contextualization module can be placed in the overall system, which are embedded on the device, in a remote server or distributed:

- Embedded on the device: The full context-aware process runs on the mobile device.
- On a server or elsewhere: All of the context-aware processing is performed on an external server that returns the identified context state to the application.
- Distributed: The architecture is scattered over different devices of the mobile system that must be connected to each other to process and infer context.

There are two main development approaches that affect the reusability and scalability of the contextualization functionality, which are the modular and ad hoc methods:

- Modular: The context module is generically prepared for any contextual application and can include additional context signals.
- Ad hoc: The context module is developed adapted to the specific application that uses it and is closed to inclusion of additional context signals.

We analyze the different degrees to which the architecture normalizes context data captured through external or internal sensors, which are by using those data directly, removing the noise from them or normalizing them:

- Direct usage: The data are included directly in the contextualization engine without a specific noise cleaning process.
- Normalize data: Post-processing techniques are used to work on relative values.

There are two main approaches to context prediction that the architecture can have, which are to infer either the current context or the future context:

- Current context: The sensed context variables are used to infer context.
- Future context: The predicted upcoming context variables are used to infer context.

We also consider if the existing architecture and models approaches are focused on the design of the system or on the result of context awareness in a real-world scenario, as in the example approach.

- Use-case focus: The approach uses real data to demonstrate how using context would work when implementing the model.
- System focus: The design is an in-depth theoretical description that might also contain an implementation but that does not use real-world contextual data to develop the application.

In Table 3, we present a summary of the taxonomy of the different architecture approaches.

	Architecture design	Mobile oriented	Functionality placement	Reusable/ scalable	Signal normalization	Context prediction	Usage example
Sorensen et al *Ad-hoc context*	Middleware	No	Device	Modular	Direct	Current	Use case
Henriscksen et al *CML*	Layer	No	Elsewhere	Modular	Post-processing	Current	Use case
Chen et al *Solar*	Middleware	No	Distributed	Modular	Direct	Current	System
Meyer et al *Context home*	Mixed	No	Distributed	Modular	Direct	Current	System
Lassila et al *Semantic*	Middleware	Yes	Distributed	Integrated	Post-processing	Current	System
Schmidt et al *Context sensing*	Layer	Yes	Device or Distributed	Modular	Post-processing	Current	Use case
Raento et al *Context Phone*	Layer	Yes	Device	Integrated	Post-processing	Current	Use case
Hofer et al *Hydrogen*	Layer	Yes	Device	Modular	Direct	Current	Use case
Fahy et al *CASS*	Middleware	Yes	Distributed	Integrated	Post-processing	Current	Use case
Chan et al *MobiPads*	Middleware	Yes	Distributed	Modular	Direct	Current	System
Mayrhofer et al *user behavior*	Layer	Yes	Distributed	Integrated	Direct	Future	Use case

Table 3 Architecture taxonomy

Overall, we observe that there are some gaps in the proposed architecture designs for mobile context-aware systems:

- **Flexibility issues:** A flexible architecture should have interfaces that allow one to easily include new context sources in a standard way.
- **Light design**: A light on-line machine learning algorithm should run smoothly on a mobile platform with limited processing power, multi-tasking, and memory and battery consumption.
- **Time-frame limitations**: All of the architectures intend to guess context at the time when the prediction is taking place and do not predict the context at future times.
- **Reusability of the context information-processing unit:** This unit transforms raw context and normalizes it to create a context signal vector that can be used by applications independently.
- **Real-world data usage:** An architecture should allow experimental test that will evaluate its performance in real cases.

In our architecture proposal, we will address the exposed limitations and the existing architecture taxonomies and consider them as design premises.

2.6 Conclusions and discussion

Context-aware computing enables the targeting and personalization of applications through contextual information. However, making context-aware applications a reality poses considerable challenges for the current schemes of development and implementation of mobile applications. It is necessary to confront the heterogeneous nature of contextual information scattered among many distinct and very different sources, the fragmentation of platforms, and a number of technological elements still under development. In addition, data acquisition, modeling and processing are rather different for context-aware applications than for regular mobile applications because they blend physical with digital data in a ubiquitous environment that requires on-line personalized responses, should be respectful of privacy concerns and should keep the users in control.

In this chapter, we have reviewed the current literature related to context-awareness: definition of context, context-aware systems foundations, existing applications, context model and prediction and the underlying architectures, among other properties.

The existing definitions of context take different approaches oriented to the understanding of data for software development: knowledge, cognition and semantic. All three approaches are focused on the grammar connotation that context but none of them are specific to mobility.

We analyzed the foundations of context-aware computing, which are the systems components and how context is used for contextualization. We find an ecosystem that is still immature made of heterogeneous elements, each with its own hardware and software characteristics that need to interact with each other. Moreover, the context sources and data also have heterogeneous formats. There is no homogeneous standard model defined to combine data and check its quality as well to allow its proper usage.

We review some examples of commercial and pilot solutions running on mobile devices of different domains (transportation, health care, automotive, academics, gaming, search engines, social networking and usability), and realize that most of them make use of context variables individually and do not combine them to create richer context scenarios.

We evaluate different current architecture approaches, and discover that they mainly focus on data acquisition from sensors, modeling and reasoning processing. Emphasis is made on the hardware architecture, the type of software approach, the knowledge taxonomy, the processing economy and efficiency, and the programming simplicity. All of the designs are ad hoc and consider as given facts the locations of contextual sources, whether the sensors are local (part of the device) or remote, the amount of users of the system (one user or many) and the type of device on which the application runs. These assumptions restrict the systems' scalability and performance. Context aware systems would need to state how to handle context changes over time or how new services can be implemented. It is also necessary to explain how to address the inherent restrictions of a mobile domain platform's

nature: the platform highly personal and always on, anywhere and anytime access and real-time responses are required, the system must process activities that are concurrent or have pauses between them, the possible ambiguity of interpretation of context due to limitations of data capture must be accounted for, and the technical limitations of the platform itself (limited battery life, low processing power, and potential unavailability of or unaffordable connectivity to the network) must be considered. All of these factors imply the need for a new architecture and tools oriented to capture context and process it that is suited to the particularities of the domain and able to evolve with it.

From our analysis, we identify two main inherent challenges that arise when developing an application that need to be taken into account when developing a context-aware application. One is the need to define a unified context model that allows capturing different context signals and smoothing them to make them homogeneous to facilitate their use. The other is the need for a combination of context reasoning techniques that allow prediction on top of context inference. In both cases, a modular and scalable system is needed.

3 Background and tools to be used

The objective of this chapter is to analyze and describe the experimental methodology that we will use throughout our thesis for context inference and prediction.

Ever since artificial intelligence was first considered a discipline in the mid-1950s, machine learning has been a central area of research. Two main reasons are that learning can potentially be used to build high-performance systems and because learning is tightly related to understanding intelligence (Quinlan 1986). Machine learning is now a burgeoning technology with a wide range of applications that has the potential to be embedded as one of the key components of intelligent systems (Holmes, Donkin et al. 1994). A major focus of machine learning research is to recognize activities, which is a topic that concerns the process of human learning. Specifically, we wish to determine how it is possible for us to learn so that we can tell a computer-enabled device how to learn and to improve from experience. We want to know how the learning process works (Arthur 2007).

Almost none of the existing context research in academia has used real-world data from mobile devices to test the performance of prediction algorithms because of the lack of traces (Song, Kotz et al. 2006), and we consider this to be crucial to understand how context-awareness really works; therefore, we will use real data available in the Reality Mining project.

In this chapter, we review the main concepts behind machine learning and explain each of the algorithms that we will use throughout the thesis to test our datasets and predict context. We also describe in detail the Reality Mining project dataset that we use to implement and test the proposal of this dissertation.

3.1 The underlying concepts of machine learning

In this section, we provide the definitions and background relevant to machine learning that will allow us to understand the tools we will use in our proposed model to predict context in mobility.

Machine learning is related to the artificial intelligence field, and it is a natural outgrowth of the intersection of computer science, statistics and cognitive science; therefore, the definitions will come from all of those fields. Machine learning is concerned with applications that improve through experience; therefore, we will need to gain some understanding of learning, machine learning, prediction, modeling and learning algorithms. The formal definition of machine learning states that “a computer program is said to learn

from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E" (Mitchell 2006).

We define all of the underlying concepts that are needed to understand the fundamental structure of learning problems.

3.1.1 Learning

The word "learning" has many different meanings. Examples are **"**knowledge or skill acquired by instruction or study" and "modification of a behavioral tendency by experience (as exposure to conditioning)" (Merriam-Webster 2010). The word can be used to describe the act of memorizing something or learning facts through analysis, practice and organizing knowledge.

Learning methods differ both in terms of the types of hypotheses they work with and the algorithms they use to find a good hypothesis given the data. The goal of learning is to find a hypothesis, h, that describes the relationship between the inputs and the outputs well. We have a set of hypotheses, and our learning algorithm must choose one. For all learning methods, we have a training dataset that we use as our basis for learning and producing an output or answer. Learning models are widely implemented for prediction of system behavior and forecasting future trends (Andreeva, Dimitrova et al. 2004).

One of the most common ways of learning is the acquisition of information with the goal of making predictions about the future. How can we predict the future? Lots of philosophers have thought about this problem. David Hume first framed the problem of induction as follows: "Inductive reasoning is the process that leads us to make generalizations from observing a number of similar cases". He claimed that all human knowledge is based on relations amongst ideas. One phenomenon not guaranteed by experience is the regular succession of events, and from observing a regularity we cannot rule out the possibility that something different might happen next time (Stoke 2003).

3.1.2 The learning process

We can visualize learning as if we are trying to find the definition of a function given a group of examples of its input and output. We have a set of training examples, which are composed of x input variables (or features) and y outputs or targets. We feed our training set into the algorithm and use a hypothesis (h) to map x to y (see Figure 4).

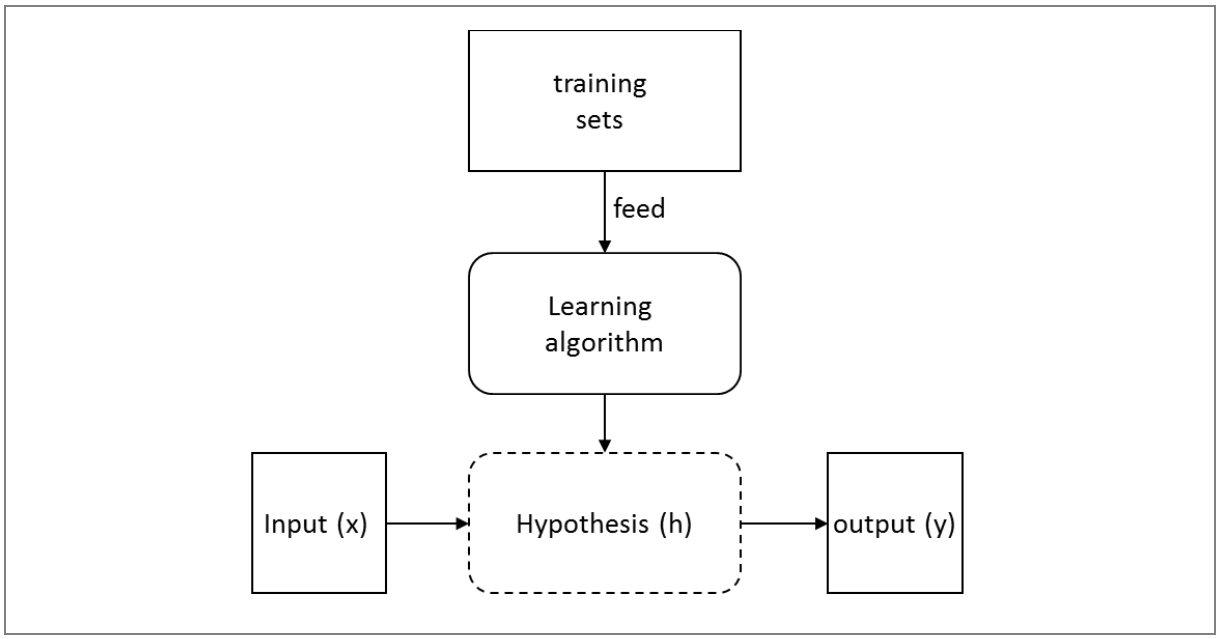

Figure 4 Structure of a learning method and its notation

Steps to follow for learning:

1. Choosing the type of training example: This step should specify the input data for our algorithm.
2. Gathering a training set: The set should be characteristic of the real world and based on measurements.
3. Determining the input feature representation of the learned function: The input is transformed into a feature vector, which contains a set of features that are descriptive of the object and are a good representation of what we want the algorithm to learn. It should contain enough information for the algorithm to learn.
4. Defining the structure of the learned function and learning algorithm: The performance of the learning algorithm is adjusted and optimized using a validation set (a subset of training set).

3.1.3 Types of learning

There are three primary methods through which we can learn: supervised, unsupervised and reinforcement.

- **Supervised:** Learn from labeled examples.

  Given a set of input and the corresponding desired output, predict outputs for future inputs, i.e., find a rule that does a good job of predicting the output associated with a new input. Examples are classification (learning to predict class labels), regression (learn to predict real-valued outputs) and time-series prediction.

- **Unsupervised:** Learn from unlabeled examples.

  Given only inputs, automatically discover representations, features, and structure, among other aspects of the input. Examples include grouping the set of examples into natural clusters and compression, among others. Clustering is to group input patterns according to clusters or natural groupings. The question is what defines "natural grouping" and how many clusters should be used. These algorithms are useful for image processing, market segmentation, and social network relations.

- **Reinforcement:** Trial and error learning.

  Given sequences of inputs, actions from a fixed set, and rewards/punishments, learn to select action sequences in a way that maximizes the expected reward. An agent interacting with the world makes observations, takes actions and is rewarded when it has learned correctly and punished when not. The learning agent performs a sequence of actions. A critic provides an evaluation for the outcome of this sequence.

We can invent new ways of learning by combination, such as semi-supervised learning (which combines both labeled and unlabeled examples to generate a classifier) or transduction (which tries to predict new outputs using training inputs, training outputs and test inputs).

3.1.4 Challenges

One of the existing challenges we encounter is that learning and adaptation should occur on-line without explicit training phases and that the user intervention should be kept to a minimum so that it is non-intrusive (Mayrhofer, Radi et al. 2003). In a mobile device usage environment, some of the major limitations are the ubiquitous nature of mobility, the real-time response requirements and the technical limitations of these types of platforms.

Another challenge is the representation of input patterns; that is, how to sense and model data:

- Feature extraction: Feature selection and preprocessing of relevant input features.
- Models for the data: Representation capacity. Common problems are that the model is too restrictive (e.g., parametric models) or the model is too flexible (over-fitting).

The most common challenge is noise, which refers to non-systematic errors on values of the attributes determined through measurements or subjective judgments. Algorithms must be able to work with inadequate data because noise can cause even the most comprehensive set of attributes to appear inadequate; however, it is counterproductive to eliminate noise from the attribute information in the training set if these same attributes will be subject to high noise (Quinlan 1986).

3.2 Machine learning algorithms

In this section, we describe the different types of machine learning algorithms that we will use throughout this research for context inference/prediction and explain how they work based on the mobile contextual data we have.

There are two main types of algorithms that we use in our research: classifiers and on-line. The main difference between them is that classifier algorithms take a set of different context data variables values at a given point of time as inputs and derive a class (context state) from them (Friedman and GoldSzmidt 1996) , while the on-line algorithms take the history of values of a single context data variable and forecast the next one to come (Katsaros and Manolopoulos 2005). We use the first for context inference and the later for context prediction, formally defining the two types of algorithms as follows:

Classifiers are functions that allows us to derive the value of a class c based on an input training data of labeled instances A with the format $[a_1, a_2, \ldots, a_n]$ (Friedman and GoldSzmidt 1996) (Hsu, Chang et al. 2003), where a_i are the features (also called attributes).

Definition 1 Classifiers

On-line predictors are based on stochastic process X that given a sequence of observed symbols $[x_1, x_2, \ldots x_n]$, predicts the next symbol x_{n+1}, (Song, Kotz et al. 2006), where x_i is the observed symbol at a given point of time.

Definition 2 On-line predictors

For clarification purposes, we define the notation that we will use for the machine learning algorithms throughout the dissertation in Table 4.

Terms	Definition
Attribute/Variable/Feature	Phone and context related signals such as active, application in use, etc.
Class/Label	Context state that we want to derive. A combination of attributes belongs to a class.
Instance/Attribute vector	A specific set of attributes with their corresponding values.
Dataset	A combination of instances which are used as input for the algorithms.
Training set	A combination of instances with known labels (classes).
Test set	One or more instances for which we want to derive the class they belong to.
Overfitting	Modelling error that occurs when a function is too closely fit to a limited set of data.

Table 4 Machine learning notation

3.2.1 Description of the algorithms

There are many ways to represent the classifier function and model the stochastic processes based on different algorithms. We select a representative sample of different machine

learning algorithms used in the literature that are very popular and have demonstrated good guess rate results for classification used in for medical diagnosis of a medical condition from symptoms, for decision making and prediction, for atmospheric predictions of severe thunderstorms for example or to create accurate user profiles interpreting user information (Quinlan 1986) (Andreeva, Dimitrova et al. 2004) (Cufoglu, Lohi et al. 2008), and for on-line prediction used for data compression, biological sequence analysis, speech and language modeling, for activity recognition or location prediction (Gopalratnam and Cook 2003) (Begleiter, El-Yaniv et al. 2004) (Kim, Helal et al. 2010) (Sigg, Haseloff et al. 2010).

We select and describe four types of classifiers of different natures: **rule-based**, **Bayes-based**, **instance-based**, and **linear function-based**, as well as **Markov Models** for on-line predictors.

Rule-based classifiers

Rule-based classifiers consist of system made of if-then rules, facts, and an interpreter that controls the application of these rules, given the facts. These rules are used to formulate the conditional statements that comprise the complete knowledge base, and they each assume the form of "`if x is A then y is B`", where `x` is the attribute, `A` is the value it takes, `y` is the class, and B is the class name. The part of the rule before "`then`" is called the premise, and the part after is termed the conclusion (Abraham 2005). These algorithms can be represented as trees or tables.

- On the **tree C4.5** algorithm case, each path from the root to one of its leaves can be transformed into a rule simply by "conjoining the tests along the path to form the antecedent part, and taking the leaf's class prediction as the class value" (Rokach and Maimon 2005).
- The **decision table** algorithms maps the set of features that are included in the table to the labeled instances from the space defined by the features (Kohavi 1995).

Bayes-based classifiers

Bayes classifiers apply Bayes 'rule (Equation 1) to compute the probability of an attribute to belong to a class on which a set of attributes E will be classified into a class C given the labeled class of the dataset with the maximum probability (Friedman and GoldSzmidt 1996). It assumes that all features are independent of each other (Cheng and Greiner 1999).

$$Probability\ (C|E) = Probability(C) * Probability(E|C)$$

Equation 1 Bayes' rule

Where C is the class label and E is the set of attributes.

Two popular Bayes classifier algorithms are Bayesian Networks and Naïve Bayes.

- **Bayesian networks** algorithms are directed acyclic graphs with a conditional probability distribution for each node. Each vertex in the graph represents a random variable, and the edges are the direct correlations between the variables (Cheng and Greiner 1999).
- On **Naïve Bayes** algorithms also uses graphs with a simple structure that has the classification node as the parent node of all other nodes which have no further connections (Cheng and Greiner 1999), as we can see in Figure 5.

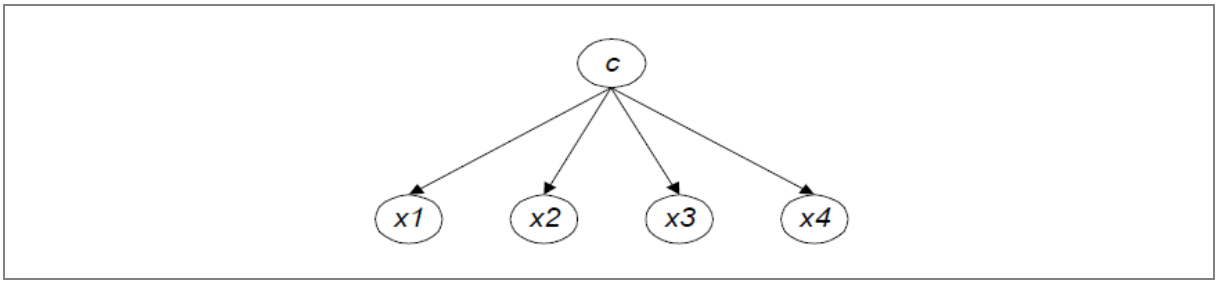

Figure 5 Naïve Bayes structure

Where C is the class and X_1, X_2, X_3, and X_4 are the features (attributes) of the dataset that is being analyzed by the classifier. It assumes independence among child nodes in the context of their parent and each features can only be related to one other feature (Friedman and GoldSzmidt 1996).

Instance-based classifiers

Instance-based classifiers output is a concept description function that maps instances (attributes pairs) to categories. An instance-based concept description includes a set of stored instances and it can also contain some information concerning their past performances during classification such as the number of correct and incorrect classification predictions.

The simplest example of Instance-based classifier is the **IB1** which uses the similarity function described in Equation 2 to find matches between a training instance and a given test instance (Aha, Kibler et al. 1991) by calculating the distance between the instances (which are attribute pairs).

$$Similiarity\ (x, y) = \sqrt{\sum_{i=1}^{n} f\left(x_i, y_{i,}\right)}$$

Equation 2 Similarity function

Where x and y are instances (attributes value pairs), n is the number of attributes and the function $f(x_i, y_i)$ is equal to $(x_i - y_i)^2$ for numeric-valued attributes and is equal to $(x_i \neq y_i)$ for Boolean and symbolic-valued attributes (Aha, Kibler et al. 1991).

Linear function-based classifiers

Function classifiers are applied to data that is linearly separable in the feature space into two different classes, and their goal is to find the margin hyper plane that separates data points based on which class they will belong to. There are many hyper planes that might classify the data, and the best choice is the one that represents the largest separation between the two classes (Cristianini and Shawe-Taylor 2000).

An example of linear function classifiers are **Support Vector Machines (SVM)**, that work by taking a set of input data and predict which of two possible classes can it belong to, a data point is viewed as an n-dimensional vector (a list of n numbers), and we want to know whether we can separate such points with a $(n - 1)$ dimensional hyper plane. The hyper plane is built using the function described in Equation 3 (Cristianini and Shawe-Taylor 2000).

$$f(x) = \sum_{i=1}^{n} (w_i x_i) + b$$

Equation 3 Linear classification formula

Where x is the attribute, x_i is the ith instance, w is the weight, and b is the bias.

Figure 6 shows a representation of how the algorithm works, the points in the graph are the attributes and are colored according to the class they belong to (Hsu, Chang et al. 2003).

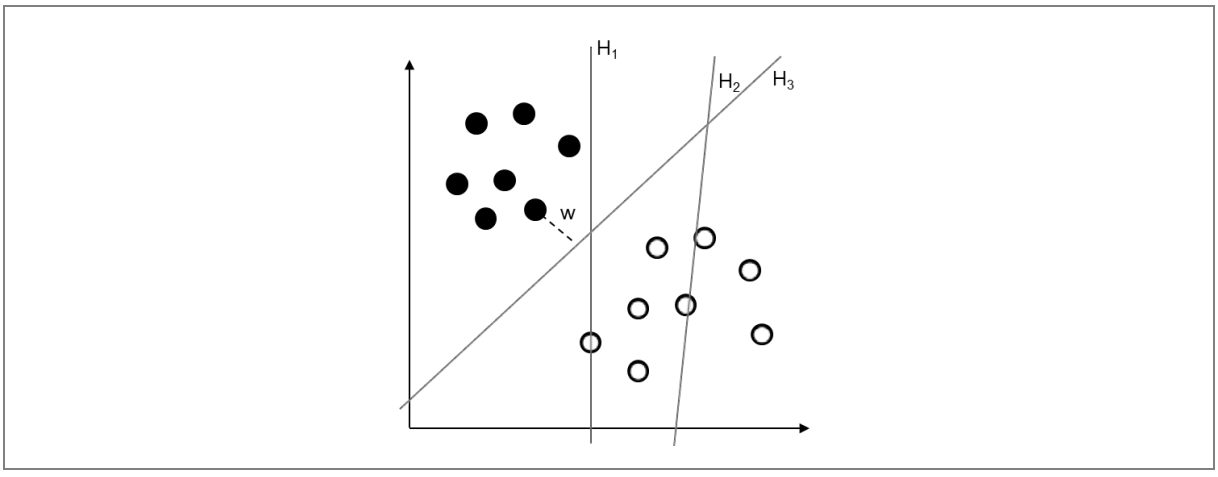

Figure 6 Representation of a Support Vector Machine linear function

Where H_1, H_2, and H_3 are three different hyper planes and w represents the distance that separates the points from the hyper plane. The optimum would be H_3.

Markov Models on-line predictors

For any sequence of events that can be modeled as a stochastic process, this algorithm employs the power of Markov models to optimally predict the next symbol in any stochastic sequence (Das, Cook et al. 2002).

The algorithm first builds a tree that keeps track of the different symbols and their number of occurrences. Then, it calculates the escape counts, probabilities and smoothing of unobserved events. The learner is given a training sequence of symbols, and based on a context, it predicts the next symbol. For any context from an alphabet, it should generate a conditional probability distribution to predict a symbol a based on the context `ctx`.

Each context is considered as a set of a finite state machine. Moving from one state to the other is calculated using a conditioned probability P that divides the number of times the symbol analyzed follows the observed context `ctx` in the symbol string over the number of times the context appears in the symbol string, as we can see in Equation 4.

$$P(X_n = a|ctx) = \frac{N(ctx\boldsymbol{a}, string)}{N(ctx^*, string)}$$

Equation 4 Markov model

Where,

- Xn is the next symbol,
- a is the observed symbol,
- `ctx` is the observed context
- `string` is the sequence of symbols
- `N(ctx`**`a`**`,string)` is the number of times the context the symbol analyzed (a) has been observed following the observed context (`ctx`)
- `N(ctx*,string)` is the number of times the context (`ctx`) has been observed in the input symbol string (`string`) followed by any other symbol (`*`).

3.2.2 Using the algorithms

To use the **classifiers to infer context**, we model the mobile context data into features (attributes) and define the target classes (labels) so that we can implement a classification system that will produce a model that allows deriving context based on the observed variables (Hsu, Chang et al. 2003). The observed variables will be the phone usage variables such as location, application in use, active connection, etc. The classes will be the context

states we want to derive, which we define ourselves, and it can be binary or multi-class (Cristianini and Shawe-Taylor 2000) according to the number of outputs it has.

Applying the classifier definition (Definition 1) we consider each phone context variable as an attribute of the classifier that we combine and label, so for example the combination of the features `[active=1, game=1, email=0, WiFi=1]` can be labeled as a class named "`relaxing`" which corresponds to the context (note that the value "`=1/0`" indicates that the attribute observed is on). The classifier will take as inputs these attributes and use a model to derive the context they belong to (Figure 7) using a different formula depending on the type of classifier we choose to use.

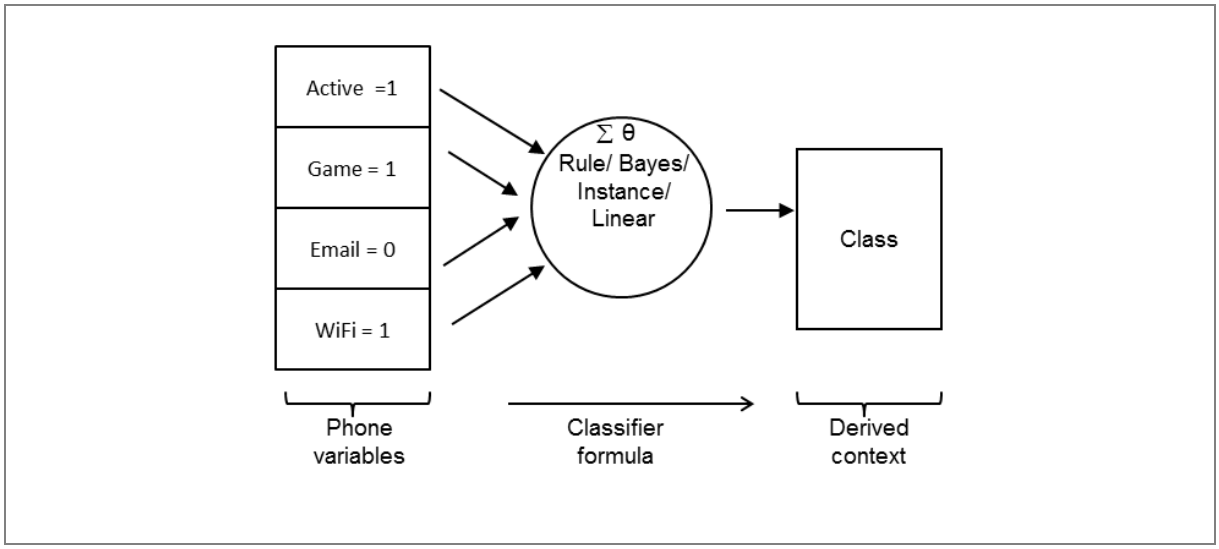

Figure 7 Example of classification algorithm usage

We illustrate how each type of classifier would derive the class using the same example and assuming that we have a dataset with some instances labeled. In Table 5 we describe the example dataset we use to describe the classifiers.

Attributes	**Training instances (labeled)**					**Test**
active	1	1	1	1	0	1
game	1	1	1	0	0	1
email	0	1	0	1	1	1
WiFi	1	0	0	0	1	1
Class label	Relaxing			Other		Unknown

Table 5 Example dataset

All algorithms create their classification model based on the training instances with the labeled classes. Each instance is an attribute vector, that is, phone variables and their values. Each algorithm has its own methodology to create the model and to classify.

Rule-based. The algorithm takes as input the attribute vector of the test instance and it starts performing tests to each attribute, depending on the outcome it moves on to perform another test and so on, recursively until it reaches the final answer in which all attributes

have the values defined on the labeled class. The combination of all the tests until the answer is reached will be the rule. The tests that the algorithm will perform are of the type of "is active = 1?" or "is email > 0". The values used to perform the tests are defined by the model and are obtained through the training set calculating the average value each attribute must have to belong to a specific class, for example the attribute's email's average value is 0.6. An example of a full rule to find the class would be "if active = 1 and game = 1 and email > 0.6 and WiFi> 0.3 → class = relaxing", therefore, when the classifier receives the test file it would apply the rule and classify the test instance as "relaxing".

Bayes-based. The algorithm creates the model by using a gauss distribution for each attribute value and calculates their mean and variance values. When deriving the class, the algorithm takes as input the instance and applies Equation 1 to calculate the probability of each class to happen based on the attribute vectors values. So for example to calculate the probability of the class of the instance to be "relaxing" it would calculate the probability of the class being "relaxing" and being "other" as we can see in Figure 8.

$$\frac{P(relaxing)p(active|relaxing)p(game|relaxing)p(email|relaxing)p(WiFi|relaxing)}{P(relaxing)p(active|relaxing)p(game|relaxing)p(email|relaxing)p(WiFi|relaxing) + P(other)p(active|other)p(game|other)p(email|other)p(WiFi|other)}$$

Figure 8 Applying Bayes rule to derive classes

Where P = a priori probability of the class (considering that the sum of probabilities of each class to happen is one).

The algorithms repeats this process to calculate the probability of the "class = other" and it would choose class with the highest probability.

When the classifier receives the test file it would apply Bayes rule as shown in Figure 8, and the test instance is classified as "relaxing" as we can see in Figure 9.

$$Prob(class = relaxing) = 1 = \frac{\frac{1}{2}\bullet\left(\frac{3}{3}\bullet\frac{3}{3}\bullet\frac{1}{3}\bullet\frac{1}{3}\right)}{\frac{1}{2}\bullet\left(\frac{3}{3}\bullet\frac{3}{3}\bullet\frac{1}{3}\bullet\frac{1}{3}\right)+\frac{1}{2}\bullet\left(\frac{1}{2}\bullet 0\bullet\frac{2}{2}\bullet\frac{1}{2}\right)} \text{ and } Prob(class = other) = 0 = \frac{\frac{1}{2}\bullet\left(\frac{1}{2}\bullet 0\bullet\frac{2}{2}\bullet\frac{1}{2}\right)}{\frac{1}{2}\bullet\left(\frac{1}{2}\bullet 0\bullet\frac{2}{2}\bullet\frac{1}{2}\right)+\frac{1}{2}\bullet\left(\frac{3}{3}\bullet\frac{3}{3}\bullet\frac{1}{3}\bullet\frac{1}{3}\right)}$$

Figure 9 Classifying the test instance using Bayes rule

Instance-based. The algorithm's output is a set of classified instances, which it calls the "Concept Description" (from now on CD). The algorithm takes as input an unlabeled instance (x), in our example [active=1,game=1,email=0,WiFi=1], reads each attribute on it and compares is with the classified instances (y), which are [active=1,game=1,email=0,WiFi=1],[active=1,game=1,email=1, WiFi=1], and [active=1,game=1,email=0,WiFi=0]. It then uses Equation 2 to calculate the similarity with each of them, it stores the resulting values into a vector named names Sim[y] that associates similarity values to classified instances and it then seeks the

maximal value in the vector. Then it selects the instance with the maximum similarity value from the `Sim[y]` vector and uses it as input to the classification function to classify the input instance`(x)` with the same class and update the CD set with the new classified instance. So `[active=1, game=1, email=1, WiFi=1]` would be classified as "relaxing" and included into the CD set.

Linear function-based. The algorithm takes as input the labeled instances and builds a model that assigns a class to the unlabeled instances. The model first maps the labeled instance's attributes as points in space, separating them by a hyper plane based on the class they belong to, as we have seen in Figure 6. It then reads the unlabeled instances' attributes and maps them into the same space and classifies them into one class or another based on which side of the hyper plane they falls into. To create the hyper plane, it makes use of Equation 3 to calculate its optimal distance to the attribute points. Figure 10 conceptually illustrates how classification using SVM would work applying it to our example over two attributes.

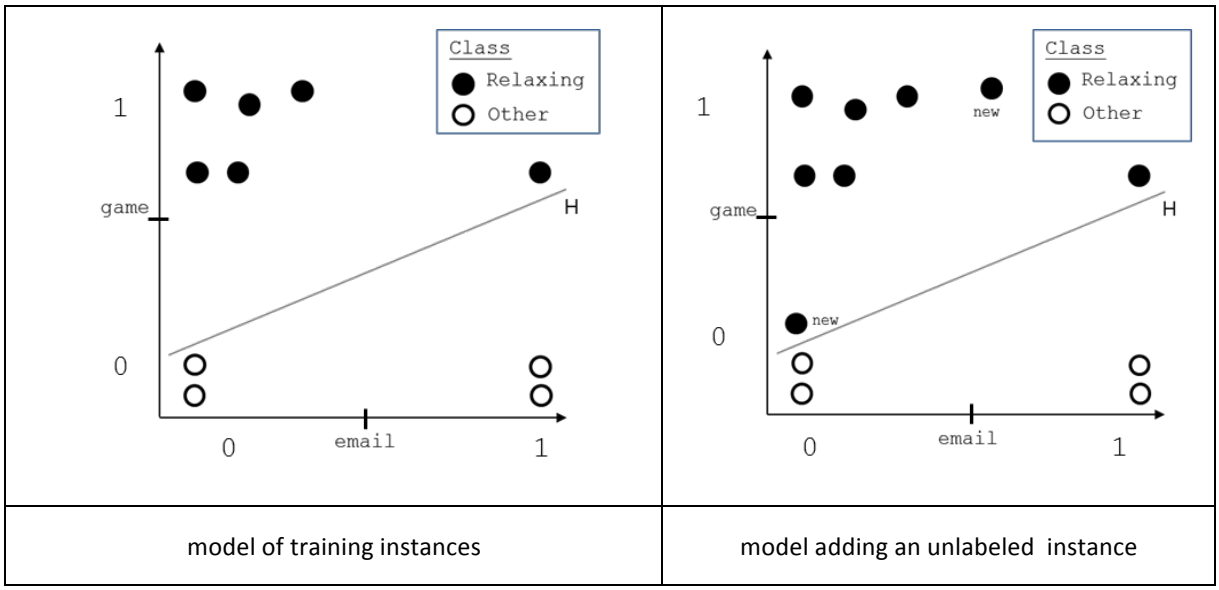

Figure 10 Conceptual example of classification using SVM for two attributes

This process is repeated recursively attribute by attribute.

To use the **Markov Models** for **on-line context prediction**, we need to model the mobile context data as symbols so that we can implement a learning system that learns from the history of events and predicts what the next event will be (Das, Cook et al. 2002). We follow a similar approach similar to the one taken in the MavHome project, which we have described in the state-of-the-art analysis, and define a sequence of observations of cell phone usage partitioning the mobile phone variables into different areas. An example of such area would be communications of the user at various times'; this will correspond to a context variable.

Applying Definition 2 On-line predictors, we consider each phone context variable as a symbol, so for example in the communication phone variable, each outgoing call can be considered as a symbol. We give a simple example of usage, based on the observed history of outgoing calls $[o_1, o_2, o_1, o_2, , o_1, o_2, o_1, o_2]$, where o_i is the outgoing call symbol to a specific number seen at time i. To calculate the next call that the user will make (o_{n+1}) we use Equation 4. So for example if we use an Order 2 algorithm, we only consider two symbols, i.e. the last two outgoing calls the user made and predict call the upcoming outgoing call (o'), we illustrate how this works in Figure 11 .

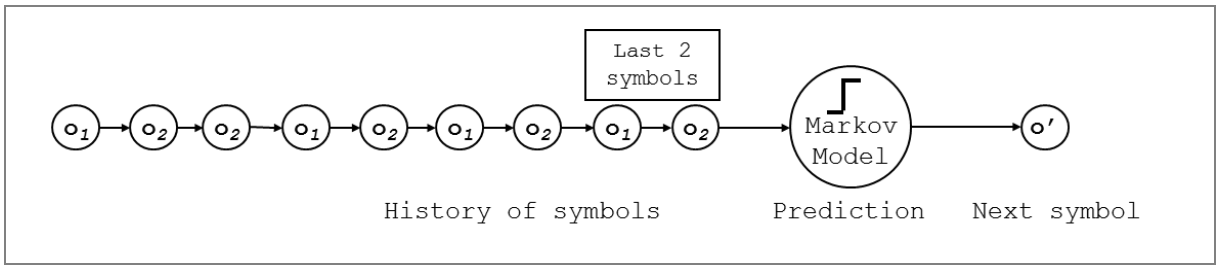

Figure 11 Example of on-line predictor usage

The context is the sequence of observed symbols in the past (o_1o_2) that are used as the basis to calculate the probability of the next symbol to appear: o', it calculates the probability of $o' = o_1$, $o' = o_2$ follows the observed sequence o_1o_2 in the history of symbols and chooses the symbol's value based on the one that had the probability of occurrence. In our example it would be symbol o_1 since it appears twice following the sequence o_1o_2, while o_2 appears only once following the sequence o_1o_2. It then reads the next symbol and notes the hit rate of its prediction.

3.3 Dataset description

In this section, we describe the dataset we have chosen to perform our tests and to exemplify and test our proposed context-aware mobile system. The dataset is the Reality Mining project, which is available at http://reality.media.mit.edu/. And stored in an SQL database and a MATLAB file. In the appendix A.5 Project selection and dataset analysis, we present the evaluation criteria used to analyze the available datasets that we have found in the academic literature and the rationale for our choice and a detailed analysis of the data.

3.3.1 Project overview

The project considers real-world datasets captured throughout the regular life of each of the study subjects and collected from the mobile devices over the period of a school year. This set represents data that are observed from everyday life, with all of the associated randomness and noise within the system inherent in real data.

The users that participate in this research are a mix of MIT Media Lab and Sloan School of Management students that have a software application that monitors context data installed on their mobile device. The study consists of 106 students, and it ran from September 2004 until June 2005 (Eagle, Pentland et al. 2009). The students enrolled in the project during the academic year of 2004/2005 and participated for a range from 19 to 311 days. The duration of the users' participation in the projects is shown in Figure 12.

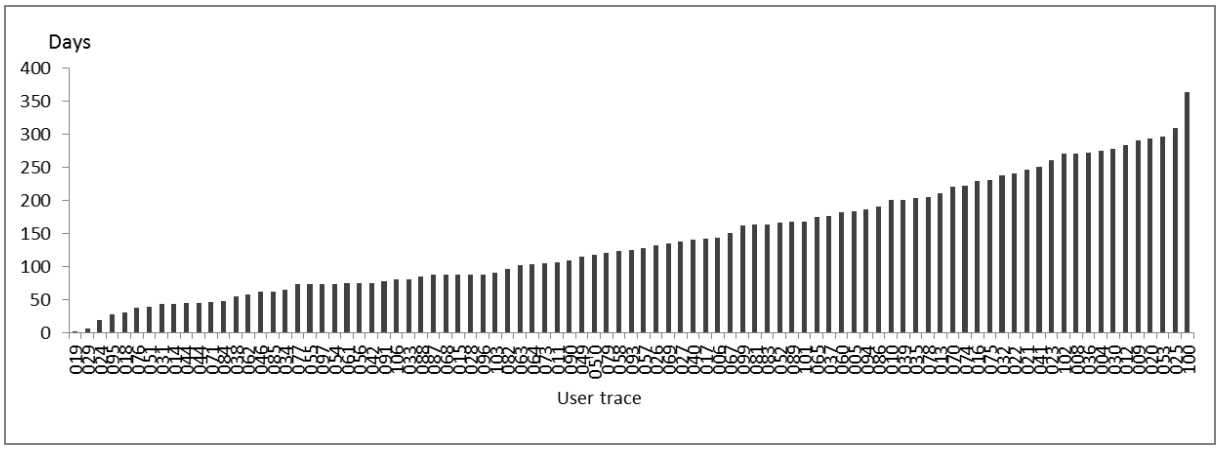

Figure 12 Number of days in the project per user

The device chosen for installation of the application was the Nokia 6600 smartphone. The captured data were kept on the smartphone's memory card, which was periodically collected by the researchers. Data were captured through the sensors available on the Nokia 6600 mobile device, which are the communication module and phone use. They capture data about cellular tower transition, Bluetooth device discovery scans and communication events.

Although the dataset is not recent, the richness of its data is still valuable today because of the large time frame of its instances, with an average of 147 days captured, and the variety of context variables covered (more than 17).

When selecting the datasets we will use in our tests, we analyze the three groups of data traces context variables (communication, location and applications used) to understand the size of the datasets and the amount and type of activity it contains, which we can see in Table 6.

	Communication related	Location's cell towers	Applications used
Average different symbols	90	919	19
Average number events	2006	24631	7922
Total traces	91	73	58

Table 6 Overview of events and symbols of the data traces

When analyzing the datasets, we observe that many contain void data, regardless of the time the user has spent on the project. So for example, user 100 has spent the longest time

in the project but there is no information recorded about its application usage, this is also the case for user 09, which has no recorded information for location.

We select those datasets that compromise trace length and degree of phone usage information, choosing the top ten users of each context variable group with the longest time-frame in the project as we show in Table 36.

Communication related			Location's cell towers			Application used			Project length	
User	Events	Symbols	User	Events	Symbols	User	Events	Symbols	User	Days
20	7750	294	4	59132	1744	20	24783	20	25	311
04	6135	100	74	58624	1774	93	16583	22	53	296
52	5689	280	12	56458	3137	36	16433	21	20	293
67	5572	160	65	56454	697	23	16220	21	93	277
40	5544	155	99	54962	2172	22	15003	19	04	277
36	4972	184	86	49597	2031	53	14955	20	12	277
53	4852	214	8	45775	2794	52	14275	20	23	277
08	4753	158	22	45546	2430	08	13035	18	60	277
81	4572	165	81	45094	1751	21	13032	22	102	275
23	4165	161	70	45035	2237	12	12628	23	36	272

Table 7 Top 10 most active and largest users

We choose the datasets that have a long amount of data and choose the ones that combine a large amount of days and that are in the top ten for two or three of the context variables:

- 10 largest communication data traces: 20, 04, 52, 67, 40, 36, 53, 08, 81 and 23.
- 10 largest location data traces: 04, 74, 12, 65, 99, 86, 08, 22, 81 and 70
- 10 largest application data traces: 20, 93, 36, 23, 22, 53, 52, 08, 21 and 12
- 10 longest data traces: 25, 53, 20, 93, 04, 12, 23, 60, 102, 36, 30, 08, and 41.

The resulting datasets to be used as input for the learning algorithm are those of users: 04, 08, 12, 20, 22, 23, 36, 53, 81 and 93.

3.3.2 Data description

The researchers created monitoring software, the ContextLog application, which was installed on the users' devices. The application captures data from phone usage, including call logs (voice or data), Bluetooth devices in proximity, cell towers, application usages and phone status.

The data available in the Reality Mining project cover many context data sources related to user interaction with the phone, including applications, calls, messages and Internet connection, and the user's location.

- **User's location**

The information is kept on two arrays with information about the antenna tower, its area and cell ID.

Location cells variable s(n).locs

This variable is an array of time-stamped tower transitions [`date, areaID.cellID` or 0 when there is no signal]

Here is an example trace:

```
732339.736053241 5188.40332
732339.737488426 5188.40811
732339.738287037 0
```

Location cell names variable s(n).cellnames

This variable is an array of `areaID.cellID` and the string the user named the location.

Trace example,

```
[ 5188.48541] 'T-MobileLogan'
[ 5188.60291] 'T-MobileSwisshouse'
[ 5187.41803] 'T-MobileAmy'
```

s(n).places

This variable represents the distribution of times the subject was at home, elsewhere, at work and with no signal.

```
Home: [24x180 double]
elsewhere: [24x180 double]
work: [24x180 double]
nosig: [24x180 double]
all: [24x180 double]
startdate: 732160
endate: 732339.753506944
hours: [4315x1 double]
dow: [4315x1 double]
cell_vec: [1x4315 cell]
places_data: [4315x1 double]
off: [1220x1 double]
starton : [42x1 double]
endon : [42x1 double]
```

- **User interaction**

This information is relative to what is happening with the phone. Is it in use? Is it charging? What application is being used? What type of communication is the user doing?

Phone charge variable s(n).charge

This variable includes the date and time the phone is charging (1) or unplugged (0). (It can be converted using datestr.)

Trace example,

```
732339.674236111 0
732339.689803241 1
732339.696423611 0
```

Application list variable s(n).apps

This variable contains the total number of times the app was used.

```
All: [1x11060 cell]
snake: 0
phone: 4430
browser: 38
camera: 92
gallery: 73
logs: 294
clock: 307
calendar: 12
video: 7
player: 5
```

Application date variable: s(n).app_dates

This variable contains the time each application was started.

```
Snake_date: []
phone_date: [4430x1 double]
browser_date: [38x1 double]
camera_date: [92x1 double]
gallery_date: [73x1 double]
logs_date: [294x1 double]
clock_date: [307x1 double]
calendar_date: [6x1 double]
video_date: [7x1 double]
player_date: [5x1 double]
```

We combine the applications with the dates into one input file that contains the date and the application that is used at that particular time.

Here is an example trace:

```
38317          Calcsoft
38237          ClockApp
38245          ConnectionMonitorUi
38294          Converter
38251          MediaGallery
```

For our analysis, we selected a subset of the applications: those that are included in the mobile device used by default for the project (Nokia 6600).

The subset of applications include the following: application manager, browser, Bluetooth connection, calculator, camera, clock, connection monitor, converter, file explorer, image viewer, logs, media gallery, menu, mixpix, notepad, phone, phonebook, pinboard, profile, screen saver, sys ap, to do, voice command and video recorder.

- Communication

The file contains all information related to communication, including communication via Internet, voice, and text.

Communication variable: s(n).comm

This variable stores all information related to communication, including messages, calls and Internet connectivity. The information is kept in a structured array with fields for each communication event. Calls to the subject's phone number are typically associated with checking voicemail, and unknown numbers are assigned a value of -1.

Here is an example trace:

```
date: 732162.65994213 –Convert using datestr
event: 299 –Unique event ID
contact: -1 –The contact ID in phone's address book? (-1 = Not
in address book)
description: 'Voice call' –Type of communication
direction: 'Outgoing' –Direction (Outgoing / Incoming)
duration: 0 –Duration in seconds (0 = didn't pick up)
hashNum: 165 –The hashed phone number of the other party
```

Aside from the log capturing, students answered a survey consisting of 25 questions related to phone usage and behavior that we will also use in our analysis to define the assumptions to use to label data. In a real life system this a priori data is not necessarily available, it is not mandatory to have it to design a system and it is enough to make some assumptions about

the phone usage that are common and flexible so that they can be applied to any type of user.

In Figure 13 we show some of the questions that the survey contains that are relevant for our research and to label attributes.

```
8. I use my phone: 1. Exclusively for work/school related matters 2. Primarily for
work/school related matters, but occasionally for personal/social use 3. Equally
for work/school and for personal/social use 4. Primarily for personal/social use 5.
Exclusively for personal/social use

10. The majority of my daily work communication is done through: (you can select
more than one) face-face discussion

11. The majority of my daily work communication is done through: (you can select
more than one) email

12. The majority of my daily work communication is done through: (you can select
more than one) phone

13. The majority of my daily work communication is done through: (you can select
more than one) text-messaging

14. The majority of my daily personal communication is done through: (you can
select more than one) face-face

discussion

15. The majority of my daily personal communication is done through: (you can
select more than one) email

16. The majority of my daily personal communication is done through: (you can
select more than one) phone

17. The majority of my daily personal communication is done through: (you can
select more than one) text-messaging
```

Figure 13 Sample of the survey data

Based on the survey data results, we obtain an insight of how each user interacts with the mobile device and their patterns,

- **User 04:** This user describes its patterns as somewhat regular and to use email mainly for personal communication and to send text messages often.
- **User 08:** This user describes its patterns as very regular and to use the phone mainly for personal communication,
- **User 12:** This user describes its patterns as somewhat regular and to use the phone both for work and personal uses, making calls, sending text and email very often for personal and for work purposes.
- **User 20:** This user describes its patterns as very regular and to use the phone both for work and personal uses, making calls, sending texts often for work and personal purposes while sending emails for work purposes.
- **User 22:** This user describes its patterns as somewhat regular and to use the phone primarily for personal uses, communicating through email for personal and for work purposes.

- **User 23:** This user describes its patterns as very regular and to use the phone both work and personal uses being very active on its usage for calls, texting and emails both personally and for work.
- **User 36:** This user describes its patterns as somewhat regular and to use the phone primarily for personal uses, sending texts quite often, and communicating face to face and by email at work and by phone and texts for personal communication.
- **User 53:** This user describes its patterns as somewhat regular and to use the phone both for work and personal uses sending texts and to communicate personally quite often face to face.
- **User 81:** This user describes its patterns as very regular and to use the phone equally for work and for personal uses, doing the majority of the work communication face to face and through email while the personal one face to face, email and phone.
- **User 93:** This user describes its patterns as not regular and to use the phone equally for work and for personal uses, sending text quite often and communicating mainly face to face for work and personally.

We will also use this information when analyzing the results of our tests, to have additional information about each user and complement the in-depth information we have on each user's context data.

3.4 Conclusions and discussion

In this chapter, we analyze in-depth the tools to be used in our thesis to make our proposal of a context-aware system that improves mobility through the inclusion of contextual information. We analyze real-world machine learning algorithms, provide the basis needed to use these types of algorithms, define the underlying concepts of machine learning and describe some of the existing algorithms. We choose four different types of classifiers of different nature to compare them:

- **Rule-based algorithms** are easy to use and understand because they simplify complex decision-making processes avoiding unnecessary computations (Safavian and Landgrebe 1991), which is crucial to a mobile environment with limited processing power.
- **Bayes-based algorithms** are robust with respect to irrelevant features (Kohavi 1995), which is necessary since the quality of the data captured by mobile sensors' is not known a priori (Leichtenstern, Luca et al. 2005) and sometimes sensor's signals are lost and the values recorded as blank (Bernardos, Tarrío et al. 2008).
- **Instance-based algorithms** are capable of handling a large class of concepts (Aha, Kibler et al. 1991), which is important to our system since it needs to processes a large and wide amount of contextual variables.

- **Linear function-based algorithms** are the easiest to apply and have shown to be very efficient (Cristianini and Shawe-Taylor 2000) (Ramana, Babu et al. 2011), which is very important for our mobile system limitations and performance needs.

We use the **Markov Model** on-line algorithm to predict future context signals using historical data as the input because they have proved to work very well in location context prediction (Song, Kotz et al. 2006).

We also describe the real life dataset that we will use throughout our thesis (the Reality Mining project), give an overview of the information it contains, describe the variables it includes and explain how we have chosen the user datasets that we will use in our tests. We select ten datasets that combine the largest amount of contextual data. We choose those datasets by ranking the datasets based on: the number of logs they have (that corresponds with the time the user has spent on the project), the amount of information each context data log has, and then blending the results.

In the next chapter, we present our context awareness system proposal. First, we define context and context awareness. Then, we present our context awareness model. Finally, we explain the underlying architecture that supports context-aware mobile applications. We present the discussion from a mobility point of view because the goal of our thesis is to contribute to improve mobility uses through context awareness. The model we define combines heterogeneous context data captured through sensors. In the architecture, we detail each of the phases needed to capture, understand and make use of context in a mobile application.

4 Context-aware modeling & architecture

There is great interest in context-aware applications that intelligently support user tasks by acting autonomously on behalf of users. The behavior of context-aware applications depends not only on their internal state and user interactions but also on the context sensed during their execution. Some early models of context information exist; however, many issues related to modeling context information have not been completely addressed. Existing models vary in the type of context information that they can represent. Although some models take into account a user's current status, such as "in a meeting", other models take into account the physical environment e.g., locations. A more generic approach to context modeling is needed to capture various features of context information, including the variety of types, dependencies, quality, and context histories. In addition, to solve the software engineering problems encountered in programming context-aware applications, appropriate abstractions, as well as scalable methods of context processing and management, are needed to support the discovery and reuse of context information.

The objective of this chapter is (1) to propose a context and a definition of context-awareness, (2) to conceptualize a model of a mobile, context-aware system that combines captured heterogeneous sensor signals into a coherent context model that can then be used by applications and, finally, (3) to define the underlying architecture that this context-aware system needs to be mobile, including limitations of these devices.

4.1 Data modeling into context

To use context in mobile services, both the definition of context and the method to process data into context must be understood. In this section, we define context and context-awareness and discuss how to aggregate data into context so that these data can be used by mobile services.

4.1.1 Context definition

To create context-aware applications, we must first understand the meaning of context from the point of view of mobility. We explore the context variables available in a mobile environment. Use of the term context, or context-awareness, usually refers to a general class of systems that can sense a continuously changing physical environment and provide relevant services to the user based on it (Dey 2001). Using this core definition, several authors (Gellersen, Schmidt et al. 2002) (Barkhuus and Dey 2003) (Lassila and Khushraj 2005)

(Coutaz and Crowley 2005) have focused on different aspects of context-awareness, such as its nature, how to model contextual data and interactions between users and context.

Based on these prior works, we propose a detailed overview of context below that satisfies the needs we have identified in the state-of-the-art analysis ("Section 2.5.2 Evaluation") of: flexibility, lightweight design appropriate for a mobile device and including time as a factor, distinguishing between current and future context.

- **Definition of context.** Context is any relevant information that can be used to characterize an entity when interacting with a user at a given point in time. In this user-centric world, an entity can be a person, device, software application, or nearly anything else. In this work, we focus primarily on mobile applications.
- **Context state.** A context state is the prominent information that characterizes how a mobile application is used, which can affect its functionality. It is the sum of multiple context variables whose combination has different meanings that can affect the mobile application. A context variable is a context signal, such as location. For example, the combination of time, date and phone usage can suggest user activity.
- **Context acquisition.** For context acquisition, we obtain a snapshot of the user's context by combining signals generated by a variety of sensors available in a mobile device: GPS, cameras, accelerometers, compass, touch, etc. (Bilton 2011). The data are analyzed and grouped into context variables that are part of our context-model (see the following section).
- **Context usage**. We use context information to enrich the mobile application user experience. For example, data captured by mobile phones can be used as inputs to improve interfaces and usability. Examples of applications are downstream LBS services, pilots on context-aware phones (Schmidt and Laerhoven 2001; Siewiorek, Smailagic et al. 2003), adaptive notepad applications (Schmidt 2000) and PDA usability (Hinkley, Pierce et al. 2005).

4.1.2 Context nature

Because it is crucial to understand the nature of context data and mobility, we describe both in detail before defining our context model framework. In their research, (Henricksen and Indulska 2004) (Henricksen, Indulska et al. 2002) make a number of observations about the nature of context information, which we explore here because it will determine the design requirements for a context model. We build on the characteristics described in both studies and add some characteristics that we have identified when analyzing real-life mobile context data.

Context information has temporal characteristics

Pervasive systems are characterized by frequent change, and, thus, the information is dynamic. However, we are interested in contextual information at a given point in time to

derive context. We can use past contextual information, which is converted into derived context states, to predict what the next context state will be.

Context information is imperfect

Because contextual data are captured through mobile devices sensors, there can be irregularities in quality or missing information. There can be a disconnect while capturing data (e.g., due to lack of coverage) that can result in incomplete or even corrupted information. Additionally, if the data are not used immediately, they may become obsolete. Therefore, it is necessary to perform a correction check and verify the completeness of the data prior to use.

Context has many alternative representations

There is usually a gap between the sensor output and the level of information needed by context-aware systems. Contextual data are usually provided in a raw format and need to be modified prior to use. This modification is not trivial because each application may require different aspects of the contextual information. Hence, multiple abstraction levels are needed for the efficient use of contextual data.

Context information is highly interrelated

There are many interrelations and dependencies between contextual data, which are not known a priori and may not be evident. Derivation rules can relate contextual data that describe how information is obtained from one or more pieces of information. For example, location data can be derived by relating cell tower IDs with contextual data for the date and time. This type of relation is a dependency because the location requires other data variables to be defined first.

Context information nature is heterogeneous

There is no unique solution for how to blend time and events to understand context changes; a logical model that includes all existing components of a context-aware, mobile computing environment does not exist. Therefore, a specific context-aggregation mechanism, which includes data classification based on the nature of the context, needs to be defined for each context-aware system.

4.2 Context-awareness model

Although data are usually unstructured and observations are assumed to be independent, there is a relation between the data points we are collecting. The sequence and subsequent observations are usually correlated. We analyze the existing context data and model them for use in our learning algorithms to infer and predict context. Our user-centric approach leads us to consider a user's interaction with a mobile device. In general, these interactions could involve any of the phone's features, such as messaging, contacts, calendar, clock, call,

web browser, etc. To build a context-aware system, we need to have an appropriate context model to represent, manipulate and access context information. By aggregating information from mobile data sources to generate contextual information, this model will help us understand how the information is related to context.

4.2.1 Main context variables

We analyze and group together context data that have the same type of contextual information. We combine contextual variables into three groups that correspond to which device-related events occur during an application execution, the physical location during the event and the time that this event occurs.

- When: The day (or week or weekend) and time frame (morning, afternoon, evening or night) are labeled as the **date and time** group (Abowd, Dey et al. 1998).
- Where: Places, such as home, work, outdoors, or an unknown place (Chen and Kotz 2000), are labeled as the **location** group (Khalil and Connelly 2006).
- What: Signal coverage, active connectivity method, battery level and physical information, e.g., the level of light or the motion of the device, such as the level of incline, speed, or device-related physical activity (Coutaz and Crowley 2005) (Khalil and Connelly 2006), are labeled as the **usage** group.

The context-grouping model presented here provides a basis for representing context variables to subsequently identify context states.

When: Date and time

It provides information on the time and date. The date and time group is an important driver because context usually consists of many partial descriptions of a situation that vary over time. It appears as a stamp next to the other context variables and is common to all variables. Here, we will use it alone as a context variable, manipulating it by partitioning a day into different time periods by which human activities are usually differentiated (i.e., morning, afternoon, evening and night slots) as shown in Figure 14 in which human activities are usually differentiated (e.g. morning slot, afternoon slot, sleep slot). We analyze each slot individually for relationships that occur between events.

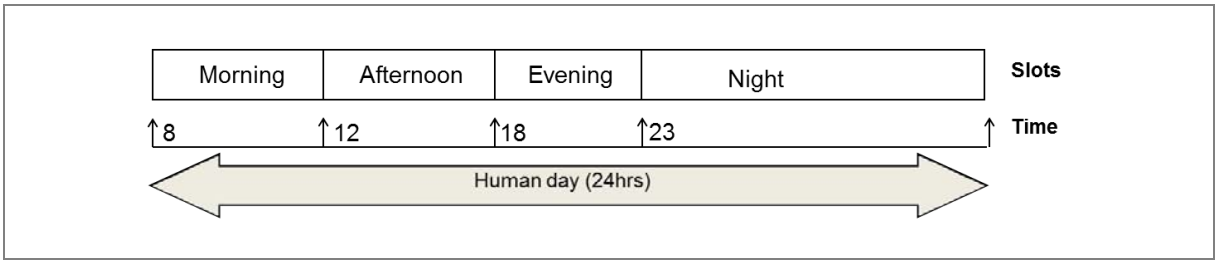

Figure 14 Day partitioning into time zones

We propose the times on which a typical day is partitioned to illustrate our point of usage of time and date variables, however, this division should be adapted to each user daily routines.

Where: Location

The location provides details of the position of the phone at a given moment; we distinguish three locations: Home (H), Work (W) and Elsewhere I. These are the main places where people can be located during the day, as indicated in Figure 15. We deploy a pure interaction-based scheme that generates an update whenever a zone boundary crossing is detected, i.e., when another location is detected.

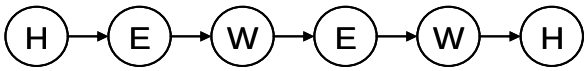

Figure 15 Sample daily itinerary

What: Usage

Usage provides information based on the status of the mobile device and its use. Status refers to whether it is on, charging or active. Use refers to user action, e.g., talking, using an application, texting and connecting to the internet.

All of these context usage variables provide streams of events that can be sampled in real time. Figure 16 and show the traces of several mobile sensors during use, such as phone usage.

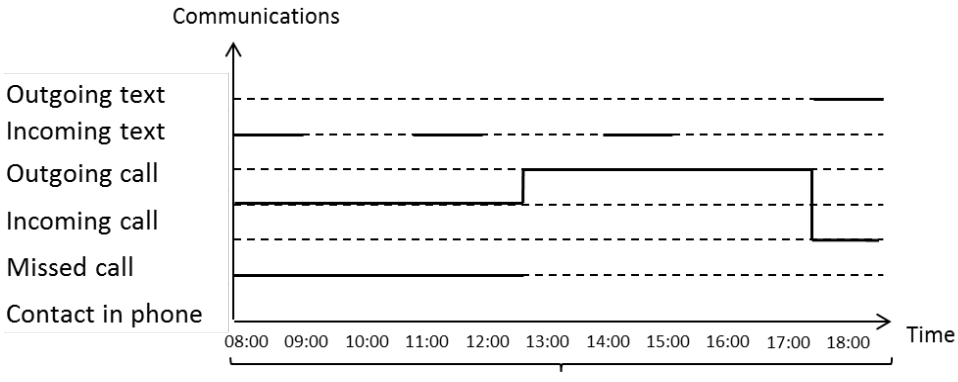

Figure 16 Typical traces of communication context signals

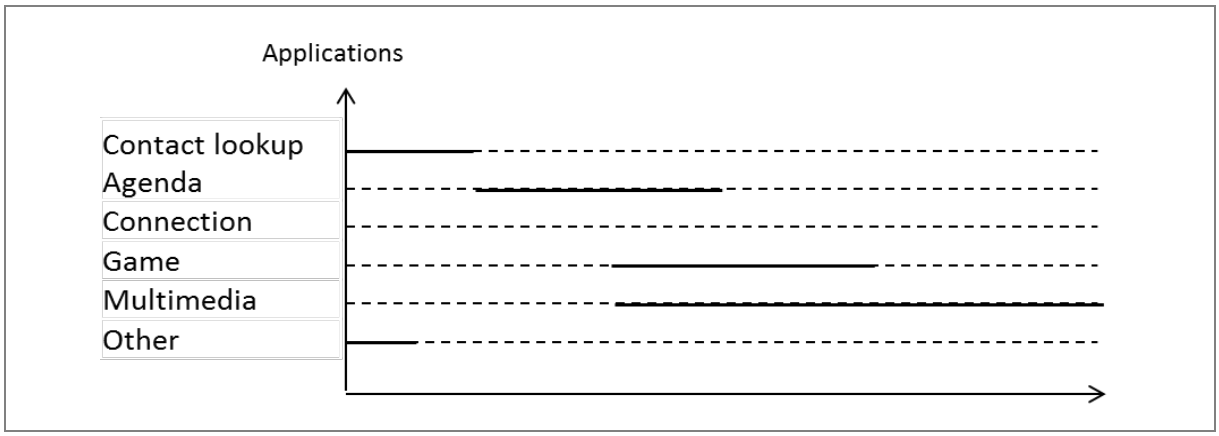

Figure 17 Typical traces of applications context signals

The data in these plots are highly heterogeneous; thus, it is challenging to make robust context inferences from these signals. We will address this challenge in the discussion of context taxonomy and states.

4.2.2 Modeling context variables

There are several common variables in mobile phones that can provide relevant information about the context of mobile applications use and that can also affect other mobile phone interactions. We model and sort them as shown in Figure 18.

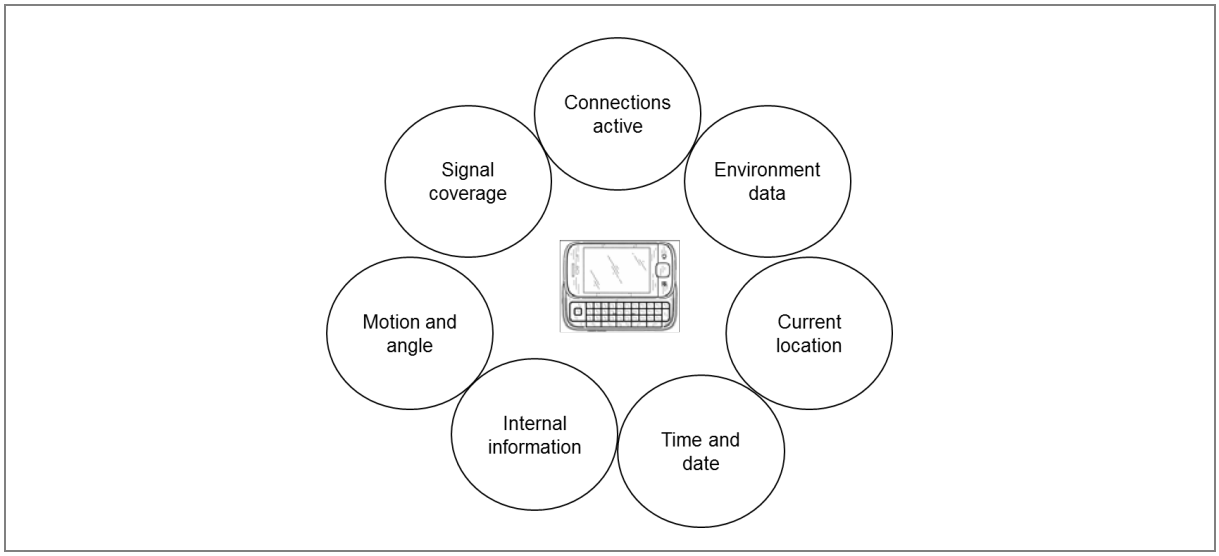

Figure 18 Context variable model

These variables provide various complementary insights into the user's interactions. For instance, if the network coverage signal is low, internet connectivity may be lost. Depending on the type of internet connection (e.g., WiFi or 3G), the speed may vary during app use. The power consumption may be affected if an application is running or if other connection methods are active, such as Bluetooth or NFC. If the battery level is low, then the mobile

device may power off. Table 8 lists several context variables and the values that they can take.

Context variables	Sub variable	Values it can take
Phone	Charging	Active or inactive
	Active	Active or inactive
	On	Active or inactive
Application	Application manager	Active or inactive
	Browser	Active or inactive
	Bluetooth	Active or inactive
	Calcsoft	Active or inactive
	Camera	Active or inactive
	Clock App	Active or inactive
	Converter	Active or inactive
	File explorer	Active or inactive
	Logs	Active or inactive
	Media Gallery	Active or inactive
	Menu	Active or inactive
	Notepad	Active or inactive
	Phone	Active or inactive
	Phonebook	Active or inactive
	Profile	Active or inactive
	SysAp	Active or inactive
	ToDo	Active or inactive
	Video recorder	Active or inactive
Internet packet	On	Active or inactive
Communication	Call outgoing and incoming	Phone number and duration
	Incoming and Outgoing	Phone number
Location	Cell code	Cell code number
	GPS coordinate	Coordinates number
Motion	Angle (X, Y, Z)	Voltage values
	Movement	Speed value

Table 8 Non-exhaustive list of mobile context variables

First, we describe our design principles; we then define our context-aware architecture and finally, detail its phases. We will take into consideration the architecture analysis and design principles identified in “Section 2.5”.

4.3 Underlying context-aware architecture

When implementing our architecture, we consider the nature of mobile devices, which are handheld computers that blend information technology, telecommunications and consumer electronics features; demand increasingly more memory, fast performance and continuous wireless connectivity; and require greater amounts of energy and processing power.

We also take a mobility point of view and address those gaps identified in the current context-awareness architectures that we analyzed in the chapter discussing the current state of the art

4.3.1 Design principles

When implementing our architecture, we take into consideration the nature of mobile devices, which are now handheld computers that blend information technology, telecommunications and consumer electronics features, demanding more memory, fast performance and continuous wireless connectivity, higher energy consumption and processing power.

When implementing our architecture, we take a mobility point of view and address all those gaps we have identified in the current context-awareness architectures we have analyzed in the state of the art chapter ("Section 2.5.2 Evaluation").

Architecture approaches

As discussed earlier, it is important to consider the limitations of mobile platforms when defining our architecture to ensure that our context classification and prediction can be performed in real-time in a memory- and processing speed-challenged environment. We define an embedded system, running without user intervention for arbitrarily long periods.

All of the architecture approaches examined in the state of the art chapter emphasize the design of an architecture that includes knowledge taxonomy and that has the highest possible processing efficiency and programming simplicity. All approaches are ad-hoc designs that assume as given fact the locations of contextual sources, whether the sensors are local (part of the device) or remote, the number of users of the system (one user or many) and the type of device in which the application runs (Baldauf, Dustdar et al. 2007). These assumptions restrict scalability and performance because the architecture needs to handle context changes over time and implementation of new services.

Moreover, existing designs do not explain how to address the restrictions inherent in a mobile domain platform: highly personal, always-on, anywhere and anytime access; need for real-time responses; processing of concurrent or intermittent activities; possible ambiguity in interpreting context due to limitations of data capture; and, lastly, technical limitations of the platform itself (limited battery life, low processing power, potential unavailability or unaffordable network connectivity). However, these requirements have not yet been addressed properly or completely in the early-stage development of most context-aware applications.

Therefore, as detailed earlier in our analysis of architecture in the state of the art chapter, we believe that new architecture designs for mobile context-aware systems should consider the following additional premises:

- a flexible architecture with interfaces, which easily incorporates new context sources, based on standard communication protocols and data definitions;
- a contextual information processing unit that performs an a priori transformation of raw context into atoms that optimize the amount of historical data needed for reasoning and that is as independent as possible both of the platform on which it runs (hardware and software) and of the type of context-aware situation; and
- a light machine learning algorithm that runs smoothly on a mobile platform (with limited processing power, memory capacity and energy constraints).

Development platforms

Currently, there are six main mobile platform OS, soon to be five when Nokia adopts Windows Phone. Each has a different strategy and development approach (Table 9).

There is no single horizontal SDK framework compatible with all of them, despite some efforts to build cross-platform development tools. There is even incompatibility within upgrade versions of the same platform, which sometimes requires a new development stage when upgrading (as when updating from Windows Mobile 6.5 to Windows Phone 7), or even when the same platform version is adopted by different handset manufacturers (as with the implementation of the Android platform by HTC and Samsung).

Obviously, the transaction costs imposed by this fragmentation are very high for interested developers, who will need to recode applications for adaptation to different platforms.

Platform	Main constituents	Development model
Apple	iPhone-iPad + OS X App Store iTunes SDK	Closed model with tight control over hardware, software and applications
Nokia	Nokia devices Ovi Symbian / SDK	Increasingly open model with control of software and hardware development
Google	Android Android marketplace Android / SDK	Open model with control of software development
RIM	Blackberry Blackberry Store RIM / SDK	Closed model with tight control over hardware, software and applications
Microsoft	Windows Marketplace Windows Phone / SDK	Closed model with tight control over software development
Linux	Linux for mobile	Open model with loose control over software development

Table 9 Mobile platform OS and application development strategy

We need to define an architecture that is flexible enough to add new context data sources and can run without exceeding the limitations of a mobile platform. Therefore, we take a platform agnostic approach and define the interfaces with the real world based on context data rather than the SDK limitations.

Hardware advances on mobile devices have enabled them to run complex applications in multitask mode. Vendors are now providing smartphones equipped with increasingly faster CPUs, more memory, dual processors, dedicated functionalities processing units, modern operating systems and sophisticated communication capacities that allow them to run network applications such as web browsing, email, and streaming media. The demand that smartphones have for multimedia features, apps, connection technologies (e.g., GPS, NFC and mobile TV) and sensor inputs (e.g., motion, inclinometers and accelerometers) requires increases in processing and memory features, which are being enabled by the growth on semiconductor market projected up to 2015. Cutting edge smartphones currently offer 32 MB (as of 2011) of embedded NAND, which is projected to grow up to 64 MB and 128 MB on-board in the upcoming models (Pulskamp 2011).

The capacities of mobile devices still lag behind those of desktop systems, and we take this into account when defining our architecture to ensure that it can properly run on a mobile device; therefore, our algorithms tests will be run on laptops with capacities similar to those of mobile devices.

4.3.2 Conceptual design

There have been several mobile, context-aware architecture designs proposed by researchers and practitioners over the last decade. Several authors (Henricksen and Indulska 2004) (Baldauf, Dustdar et al. 2007) (Schmidt and Laerhoven 2001) have proposed a layered design in which the context processing functionality is split between clearly defined layers, which then perform the three main tasks of understanding, processing and using contextual information in the application. In this approach, integration of new context sources must be propagated throughout all the layers, thus increasing the complexity of maintenance and updates.

Other authors (Hinkley, Pierce et al. 2005) (Lassila and Khushraj 2005) (Pauty, Preuveneers et al. 2006) focus on a middleware design: the transformation of data into meaningful context information occurs in a single piece of middleware software. Contextual information is captured by sensors and analyzed directly by the middleware reasoning engine. The separation between the middleware and the applications is unclear. There are two approaches for middleware: centralized in a remote server or distributed over different devices in the mobile system. In this approach, processing of context is slower because pre-processing of data does not occur. The middleware is typically appropriate to a specific context-aware situation.

We propose a generic architecture for processing context signals and making them available to downstream application conceptual diagram, in which contextual information is processed before context is derived. Our architecture design is a mix of layers and middleware. The context processing functionality is split among three different phases, through which context information is propagated, and each has a different function for understanding and processing contextual information and for identifying the context state to provide the information as an output to the application. An overview of this proposal can be observed in Figure 19.

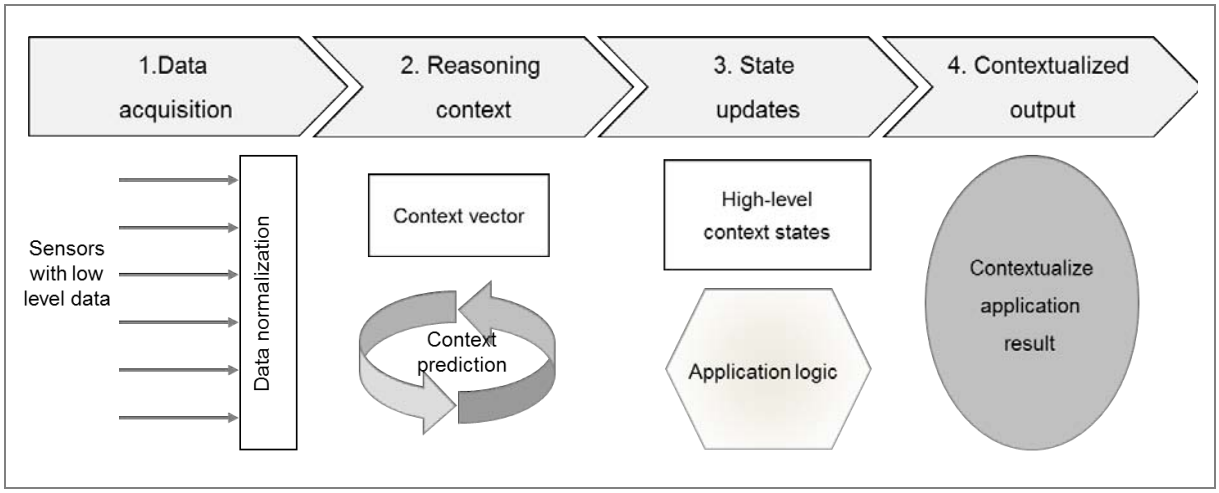

Figure 19 Context-aware architecture conceptual diagram

First, context is captured through sensors, and then it is processed in two stages: higher-level context states and data normalization. Finally, context data are ready to be used to derive and identify a context that can be used by an application. We will define each phase in detail in the following section.

We propose to wrap-up this architecture into a middleware module that is included in mobile application that we want to contextualize, as observed in Figure 20

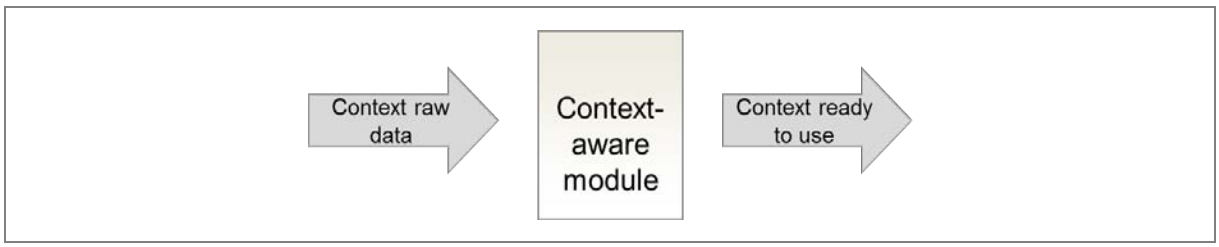

Figure 20 Context-aware middleware module

4.3.3 Architecture phases

Context signals are processed in four phases: phases 1 and 2 are application-independent and only concern the acquisition of context signals, and the modeling and prediction of

higher-level context states. Phases 3 and 4 are adapted to a mobile application that will use the context states.

Phase 1: Data acquisition

Phase 1 takes as an input the raw context sensor signals and processes them into context information logs; Figure 21 is a conceptual diagram of this process.

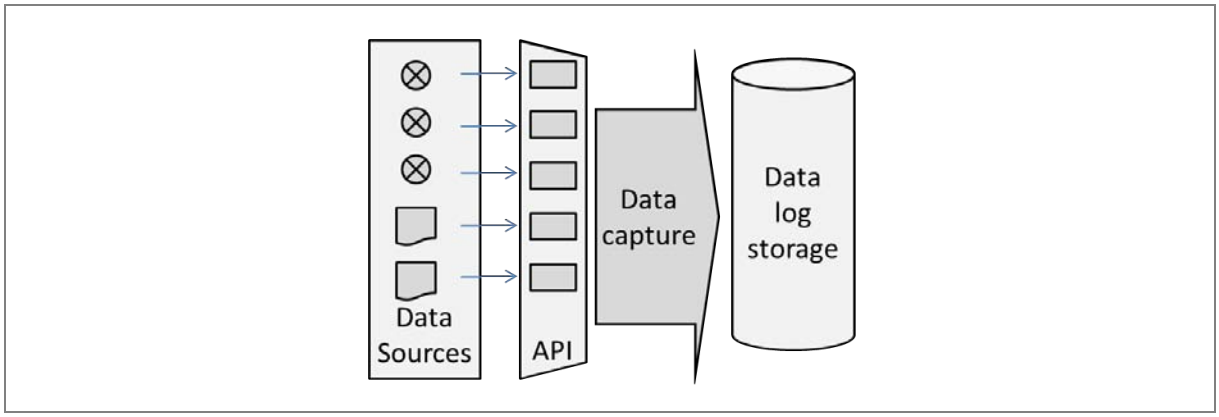

Figure 21 Conceptual diagram of sensor data logging

Context data are available to mobile devices through their sensors; a sensor is a hardware measuring device that "measures a physical quantity in its immediate vicinity and converts that quantity into small sets of numeric digital values" (Nokia 2011). Most smartphones today provide programmatic interfaces to access sensor data, thereby enabling application developers to observe the external and internal states of the system (Lassila and Khushraj 2005). Examples of common sensors found in modern smartphones are cameras, GPS receivers, microphones, accelerometers, digital compass, network connectivity interfaces and touch screens. Sensors capture contextual information from the environment and convert it into signals that can be read and transformed into meta-data. They are now routinely included as features of several portable devices; Figure 22 is a timeline of the advancements made in the sensory technology found in mobile device.

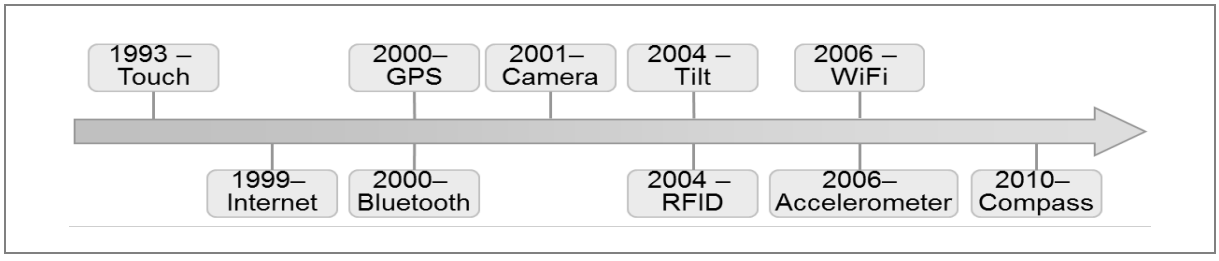

Figure 22 Sensor embedding-industry sources adapted

The availability of sensor-rich mobile devices is driven both by advances in sensor technology and by the increased use of smartphones (Albrecht Schmidt, Michael Beigl et al. 1999), (Korpipää and Mäntyjärvi 2003), (Peter Fröhlich, Antti Oulasvirta et al. 2011).

Context information can be captured through different types of sensors through the use of an API available in mobile programming SDK to access mobile device sensors that deliver their context signals.

Table 10 lists several context variables (signals), the sensors that capture them and the corresponding context variables to which they belong following the model defined in Figure 18 and Table 8.

Signal	Sensors	Description	Context variables
Temperature	Thermometer	External temperature or body heat.	Environmental data (Application)
Movement	Accelerometer	Inclination, acceleration or motion of the device.	Motion and angle (Motion)
Location	GPS	Position, location and proximity to places.	Location (Location)
Tactile interaction	Mouse Joystick Keyboard	Interaction between the user and the device, indicating its motion (right, left, middle buttons) or the key pressed.	Internal information (Application)
Audio	Microphone	External noise level, type of input (music, speaking) and base frequency.	Environmental data (Application)
Light	Camera	Light intensity or spectrum.	Environmental data (Application)
Images, video	Camera	Recognition of surrounding environment and relative position	Environmental data (Application)
Angle	Tilt angle	How the phone is being held, if it is horizontal, vertical or diagonal.	Motion and angle (Motion)
Battery level	battery charge	Phone charging and battery status (percentage remaining).	Internal information (Phone)
Time	Clock	time gauge	Time and time frame (Phone)
Ring	Profile	Phone ring mode, silent, meeting, outdoors or loud.	Internal information (Phone)
Device usage	Internal Log	Use of the mobile device, information typed, accessed, etc	Internal information (Phone)
Connection	Communications module	Indicates which network the phone is connected to: PAN, MAN, LAN or WAN.	Connections active (Internet,other)

Table 10 Context signal and sensor map

These examples were taken from the common Java API specification for Nokia (Nokia 2011) and on the research by Albrecht Schmidt and Kristof Van Laerhoven on how to build smart appliances (Schmidt and Laerhoven 2001). The API allows for data retrieval from sensors either synchronously or asynchronously, and it defines the properties of the data that it is measuring, e.g., units, scale and accuracy. The sensor can measure different properties or dimensions simultaneously, which are considered as values, and are stored in a data object to be retrieved when needed. They store the name, accuracy, data type, measurement ranges, scale and unit.

Once the context data are transformed into logs, they are processed and interpreted to normalize the data into a context vector; Figure 23 is a conceptual diagram of this process.

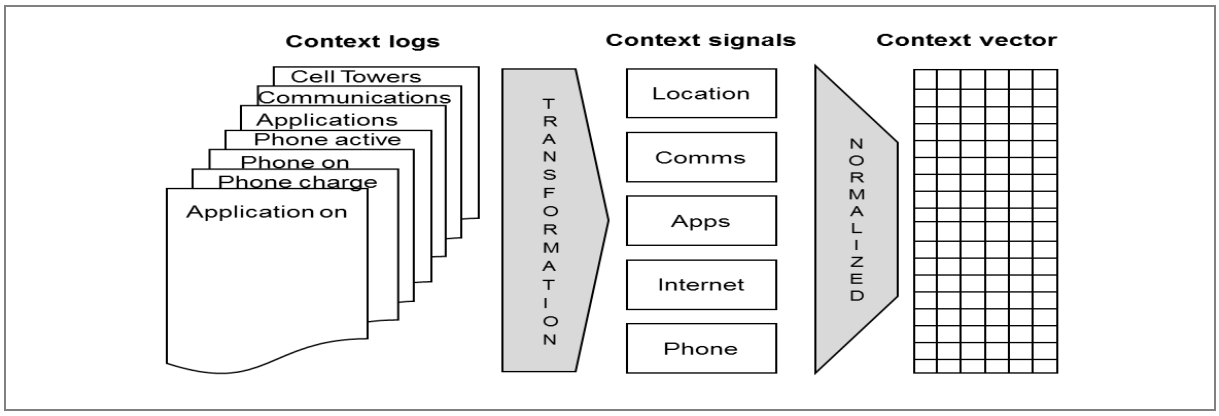

Figure 23 Example of sensor data normalization

The data are extracted from the hardware sensors and stored in logs; the data are then simplified, cleaned-up to eliminate sensor noise (e.g., null values) and formatted in such a way that they can be understood. We first transform the sensed data into context vectors that store several variables that can be interpreted better to produce a context vector with clean signals and well-defined ranges.

To do this, we apply a defined mobile context-aware model that will help us understand and transform information to context. This model is based on generic mobile usage related data; the main challenge we face is that these data are quite heterogeneous, which makes it difficult to make robust context inferences on the basis of these signals.

Therefore, we propose a framework that will allow us to unify context and allows us to combine heterogeneous sensor signals captured into a coherent context model that can be used to derive context. It creates a common structure for the data captured so that we can create a context matrix to use to derive context.

We model the context signals that affect a mobile device in a uniform way, following our earlier approach, we define variables related to when an action is happening (week day, time), where the action is happening (location, position) and usage related (phone, communications, internet connection, application in use). We show the resulting context vector in Table 11.

Week day	Time	Phone	Communication	Connectivity	Location	Position	Applications

Table 11 Context vector

When calculating the values of variables, we synchronize them with respect to time and group them into discrete time-slices, creating a context vector ***V***(t) which is parameterized by time *t*.

The duration of the time-slice determines the amount of context data available for each context reasoning step. During real-time use, the group size also determines how quickly an application can respond to a change in context variables. The content states of the vector can be calculated with Equation 5.

$$\vec{V}(t) = \{CS_1(t), CS_2(t) \dots CS_n(t)\}$$

Equation 5 Calculating context vector values

Where CS is the context signal, $CS_i(t)$ is the value of the "i" context signal (variable) for time slice t.

Certain variables cannot occur simultaneously and thus have binary values, which is the case of Weekday and Time, which will take values of either 1 or 0. For example, it is either Monday or Tuesday, and it is either Morning or Afternoon.

The remaining variables possess other values, which are calculated as the fraction of time that a specific signal is on during the time stamps of the given time-slice. For example, if there is a one hour time-slice, we would add up how many times a context signal of a particular numeric type, such as Phone On or Communication text, is on (1) and divide by the total number of time stamps recorded in the time-slice.

Phase 2: Reasoning context

Certain context variables become context states when they are combined; for example, time plus location can indicate a working activity, and motion plus phone usage can indicate a moving activity. We use the classification algorithms described earlier to infer context states, i.e., class to which the combination of context variables belongs. We need to select the context variables that become more useful when combined; for example, for time plus location, we would test several context variables by combining them and use the classifier to determine which prediction results are better, thereby indicating how relevant the variables are to the class.

We label datasets based on the target context states and create a context vector, which is taken as the input to infer the different context states; this process will be performed with machine learning techniques. Figure 24 is a conceptual diagram of this process.

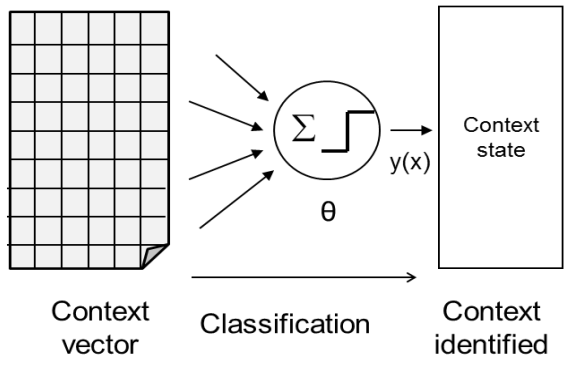

Figure 24 Context identification

We deduce the context state from the available data, which is classified into patterns that can be interpreted as states of a state machine. When the user advances from one context to another, sensor readings change, and another state becomes active, thereby reflecting the context change. Thus, interpreting context changes as a state trajectory allows for forecasting of the trajectory and prediction of the anticipated context. In our analysis, we consider context to be provided as a resource; the machine interprets the datasets from the user's context in such a way that we can use it to understand human behavior (Mika Raento, Antti Oulasvirta et al. 2005).

Furthermore, we propose an improvement to our architecture to allow it to predict future context states: the addition of a previous step in which we predict the future individual context signals using state machine models, such as Markov family algorithms. The predicted context signals will be combined into a context signal vector that is used by the classifier to infer context. In Figure 25, we can see the updated design of the context states identification process that was defined in Figure 24.

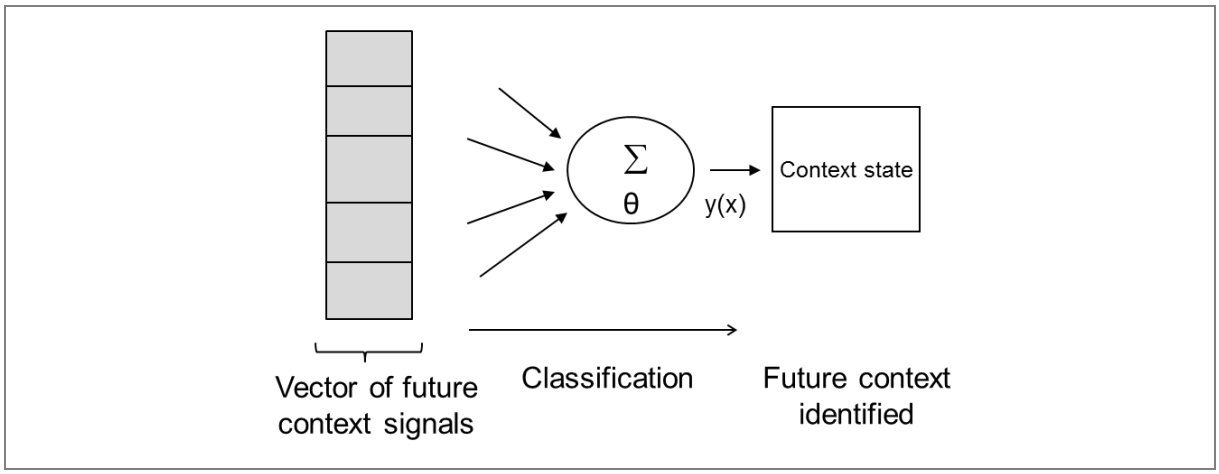

Figure 25 Prediction of future context states

When classifying context states, we continue to use the classifier model that was created after training with user data, but we input the predicted context signals to enable it to predict the upcoming context state. The rationale for this is that once we have modeled the context variables into symbols, the upcoming state can be predicted based on the historical state using an on-line algorithm, such as the Markov Model, which was previously defined in our tools analysis ("Section 3.2.1 Description of the algorithms").

This proposal solves the time frame restriction of predicting future context that we have detected in architectures in our state of the art analysis.

Methodology to follow to test machine learning algorithms

The methodology needed to use the machine learning algorithm follows the steps described in Figure 26 and is based on generic research phases (Kivi 2007) in which we first plan what

to learn and then setup a collection and analysis method for the data traces to be used for our tests.

First, we define the required data from a training set and then format them into a matrix. We filter the specific data to test and store them in a text file that will be fed to a learning algorithm that will run it and calculate its guessing percent rate.

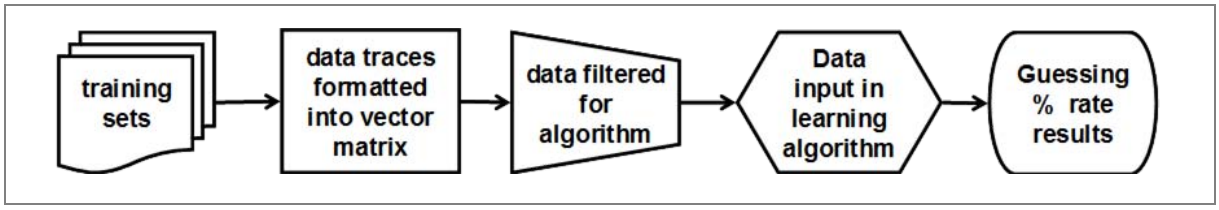

Figure 26 Test methodology

- Training sets: Analysis of datasets to understand granularity and size.
- Data traces formatted into vector matrix: raw data from the logs are formatted into context variables and stored in a matrix structure.
- Data filtered for algorithm: selection of relevant data and modeling variables into context.
- Data input in learning algorithm: Labeling of symbols and states to be used by the algorithm.
- Guessing % rate results: running of the algorithm with training data to obtain the results of the predictions.

Phases 3 and 4: State updates and contextualized output

This phase takes as an input the context states and outputs the adapted mobile application behavior, triggering a specific action associated with it; Figure 27 is a conceptual diagram of this process.

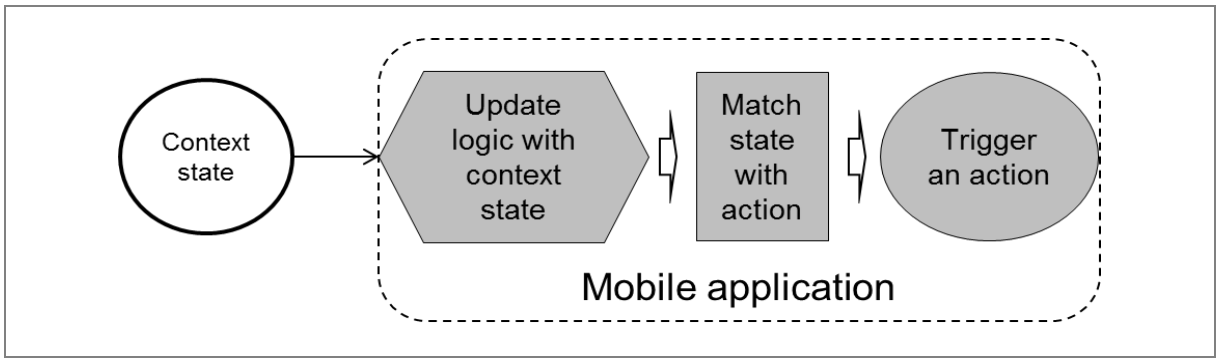

Figure 27 Updating applications with context

The context states will be driven by the nature and requirements of the application, which need not be mutually exclusive. For example, we may want to define a set of activity

contexts that a user is in as well as a number of orthogonal device states, such as “low battery”, “device location”, etc.

After having gained a detailed, robust understanding of the context state, the application can update its internal logic and produce contextualized outputs. It makes use of application-specific input/output routines that include the inferred context and that trigger an action. To enable triggering an action, a map between actions and context is defined and stored in the application, which should specify the threshold value that triggers an action and the exact time that it should occur. Some contexts may have several actions associated to them; in these cases, extra information will be used in combination with the associated probability to choose the most appropriate action.

Mobile applications are essential because the mobile domain seems to be the most suitable arena to test the concept of context-awareness. In mobile systems, changes in the user’s environment are fast, unexpected and have significant impacts on the user’s needs; reacting to the user’s context is necessary to better align with the user’s expectations. For example, the requirements of an interaction with a mobile device may change according to context: when in a business meeting, we might want our cell phone to be silent automatically so as to not disturb the meeting. However, in a social environment, we might want the volume to be turned up high so as to hear a call in what is likely a noisy environment. To support more powerful contextualization scenarios in the future, it will be necessary for mobile applications to become more aware of different context variables. In addition to user location and time, other context information, such as users’ activities and device characteristics, will help adapt applications to and personalize them for their users; this adaptation is why we propose to define context as a combination of variables.

4.4 Implementing our proposal

In this section, we illustrate how our system (architecture and model) works in a real-life example based on contextual data captured from handsets. We explain how to implement our proposed context-aware system and its architecture phases. We perform two preliminary steps: identify and analyze in-depth the context data that is available, and define the context states that will make a mobile application context-aware.

We implement our model on different devices; we describe their hardware characteristics in Table 12 (NokiaDeveloper 2003), (PCWorld 2008), (CNET 2012), (Acer 2012).

	Netbook	Laptop	Smartphone	Touch Smartphone
Vendor	Acer	Dell	Nokia	Nokia
Model	Aspire One	Latitude E4200	6600	Lumia 900
Operating system	Windows XP Home	Windows 7	Symbian S60 2nd edition	Windows Phone 7.5
Device type	Netbook	Laptop	Smartphone	Smartphone
CPU Frequency	1.6 GHz	1.4GHz	104 MHz	1.4GHz
Processor	Intel Atom processor	Mobile Intel Express Chipsets	ARM-9 series	Snapdragon (MSM8255)
Memory capacity	1 GB RAM 160 GB Hard drive	2 GB RAM 5 GB Hard drive	6 MB	512MB RAM 16 GB Hard drive

Table 12 Devices technical specifications

4.4.1 Preliminary context data analysis (Phase 1)

We analyze in detail the existing context data, a dataset from Reality Mining http://reality.media.mit.edu/ that we selected after evaluating several real-life datasets on mobile context information in the literature. Full evaluation is in Appendix A: datasets evaluation.

The goal of the Reality Mining project is to analyze social interaction and human behavior from mobile phone data. The project consisted of 106 students and ran from September 2004 until June 2005, with an average of 147 days, with a range from 19 to 311 (Eagle, Pentland et al. 2009).

The data available on the project covers many context data sources related to user interaction with the phone, such as applications, calls, messages, internet connection and location.

We choose for our tests those data traces related to location and those applications with the largest datasets and the greatest variety of **activity**; from these traces we obtain the top 10 traces: Trace04, Trace08, Trace12, Trace20, Trace22, Trace23, Trace36, Trace53, Trace81 and Trace93, shown in Table 13.

		Location		Apps	
trace	Days	Number events	Different symbols	Number events	Different symbols
93	277	80971	2691	12344	20
81	224	45094	1751	8606	19
53	296	34614	1744	12249	18
36	272	56174	2206	12869	20
23	277	72403	2840	12491	19
22	240	45546	2430	10508	18
20	293	72362	3137	20284	18
12	277	56458	3137	9838	21
08	256	45775	2794	10508	16
04	277	59132	1744	10728	19

Table 13 Selected traces

We have selected a subset of data traces that has enough information to be used as a test set and that can be modeled in a stochastic manner to serve as the input for the learning algorithm; we chose data traces longer than 225 days that appear in 2 or 3 of the context variables, as listed in Table 36.

	Communications	Locations	Applications
Average different symbols	90	919	19
Average number events	2006	24631	7922

Table 14 Overview of events and symbols of the data traces

4.4.2 Context states definition (Phase 1)

To define the context states we want to use, we must analyze the scenarios that will be our context states, o. We must define the context states we want to derive; to do this, we must analyze three main context variables groups that we have defined by relating them to each other when defining the various contexts that are specific to on-campus student life.

Step 1 – Activity analysis

Analyze the activity and the actions that compose it with a list of actions that a user can do with a mobile device, shown in Table 15.

Activity
User calls a contact
User calls another contact
User calls an un-identified number
User makes an unanswered call to an un-identified number
User misses a call form a contact

Activity
User reads an SMS from a contact
User sends an SMS to a contact
User reads an SMS from a contact
Phone is charging and off.
Phone is on.
User browses the web.
User listens to music
User watches a video
User looks at the phone's calendar
User looks up a contact in the phone book
User is commuting
User is using the GPS app
User is browsing the web

Table 15 Example of user interaction with mobile device

These activities, when combined, create a plausible scenario for mobile device usage based on the context variable values, which is called a "context stamp" (Kivi 2007). We define three main context states: **relaxing**, **working, and moving**. Each state is a context scenario for the user, highly dependent on location but also incorporating activities performed with the mobile device, as well as the time and date, as shown in Table 16 .

Context state	Definition
Relaxing	Student is doing an entertainment activity or sleeping.
Working	Student is working on university related tasks.
Moving	Student is on the move running an errand or commuting.

Table 16 Context states definition

Each activity is composed of several variables that correspond to the context signals that are on at a given time, which is the context vector defined in Equation 5.

We look for the context variables involved in the activity, which are listed in Table 17.

Activity	Usage	Location	Time frame
User calls a contact	Communications Applications	Anywhere	Morning, Afternoon, Evening
User calls another contact	Communications Applications	Anywhere	Morning, Afternoon, Evening
User calls an un-identified number	Communications	Office	Morning, Afternoon, Evening
User misses a call form a contact	Communications Applications	Office	Morning, Afternoon,
User misses a call form a contact	Communications Applications	Office	Morning, Afternoon, Evening

Activity	Usage	Location	Time frame
User reads an SMS from a contact	Communications Applications	Office/ Home/ Elsewhere	Morning, Afternoon, Evening
User sends an SMS to a contact	Communications Applications	Office/ Home/ Elsewhere	Morning, Afternoon, Evening
User reads an SMS from a contact	Communications Applications	Office/ Home/ Elsewhere	Morning, Afternoon, Evening
User sends an SMS to a contact	Communications Applications	Office/ Home/ Elsewhere	Morning, Afternoon, Evening
User reads an SMS from a contact	Communications Applications	Office/ Home/ Elsewhere	Morning, Afternoon, Evening
User sends an SMS to a contact	Communications Applications	Office/ Home/ Elsewhere	Morning, Afternoon, Evening
Phone is charging and off.	Phone charging	Office Home	Morning, Afternoon, Evening, Night
Phone is on	Phone on Phone active	Office/ Home/ Elsewhere	Morning, Afternoon, Evening, Night
User browses the web.	Internet	Elsewhere	Morning, Afternoon, Evening
User listens to music	Applications	Home/ Elsewhere	Morning, Afternoon, Evening
User watches a video	Applications	Home/ Elsewhere	Morning, Afternoon, Evening
User looks at the phone's calendar	Applications	Office/ Home/ Elsewhere	Morning, Afternoon, Evening
User looks up a contact in the phone book	Communications	Office/ Home/ Elsewhere	Morning, Afternoon, Evening

Table 17 Activity and context variables

We then define a semantic model of context states, which involves context variables; the relationship between the two is shown in Figure 28.

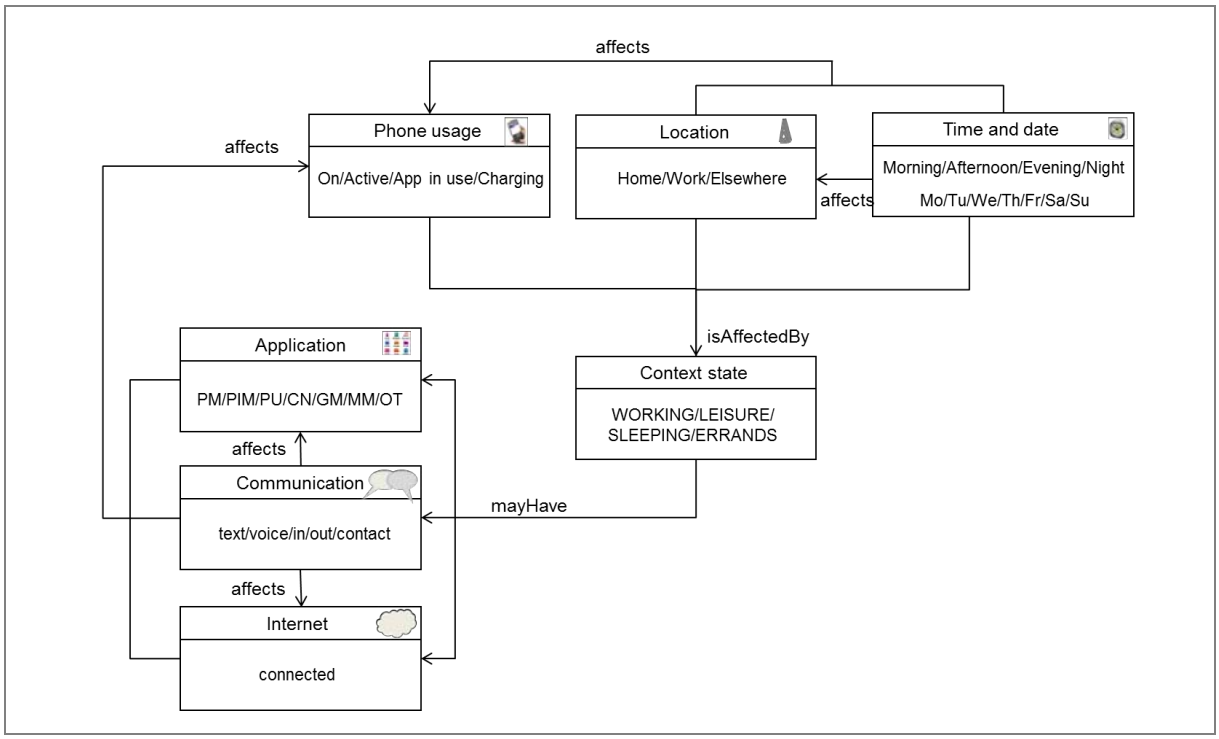

Figure 28 Context Meta model

4.4.3 Modeling context data (Phase 1)

The context-awareness model revolves around the sense-model-plan-act cycle. First, we must sense the mobile device's context (i.e., usage environment), and then we have to model it into situations to create a plan of action. Context is not only a state with a predefined environment and set of interaction resources; it is actually the result of interacting with a changing environment. Consequently, we face the challenge of understanding constantly changing variables.

We define taxonomy to represent our context model by matching context with time and date variables, which tell us what occurred within a given time slot. We select from our model those context variables available from the Reality Mining project, as shown in Figure 29.

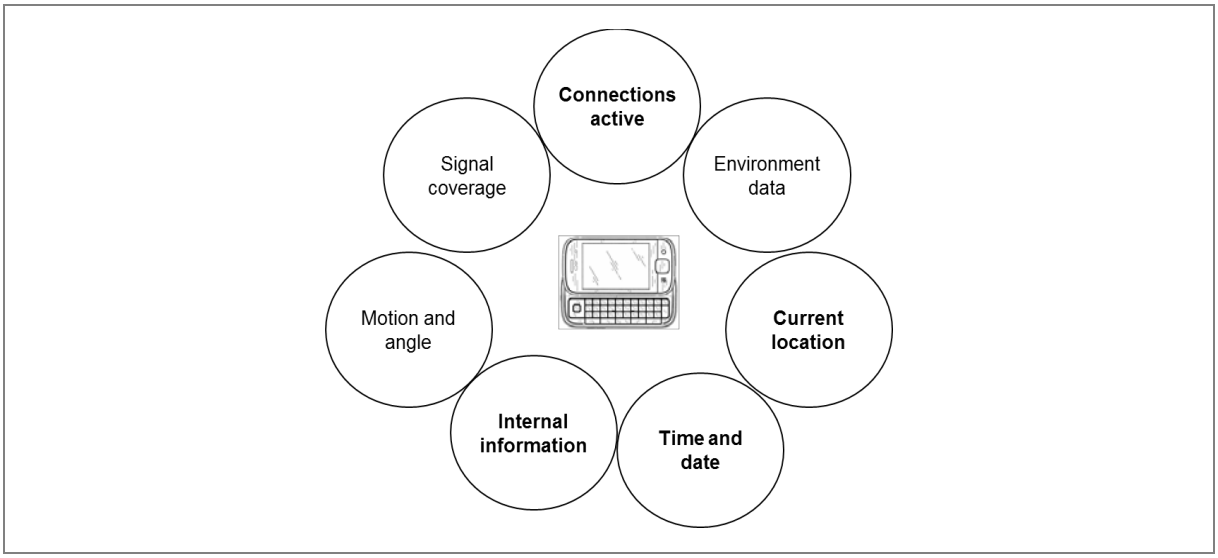

Figure 29 Available context data (highlighted in bold)

We apply the context model using the context vector $V(t)$ formula (Equation 5); the result, listed in Table 18, is based on the available data, although our model can be expanded to accommodate all available mobile context signals.

Context variables		**Description**	**Values**
Date of the week context variable	Mo	Monday	yes or no (1 or 0)
	Tu	Tuesday	
	We	Wednesday	
	Th	Thursday	
	Fr	Friday	
	Sa	Saturday	
	Su	Sunday	
Day time-slice context variable	M	Morning (8-12)	
	A	Afternoon (12-18)	
	E	Evening (18-23)	
	N	Night (23 on)	
Phone context variables	On	Phone is turned on	% the signal is on
	Ac	Phone is in use	
	App	Application is running	
	Ch	Phone is charging	
Communication and text context variables	Tx	Text message	
	Vc	Voice call	
	In	Incoming	
	Out	Outgoing	
	Tout	Outgoing text to phone number	
	Tin	Incoming text to phone number	
	Vout	Voice call outgoing to phone number	
	Vin	Voice call incoming from phone number	
	PB	Caller or recipient is in phone book	
	Mss	Missed call	
	Sec	Duration of the call in seconds	

Context variables		Description	Values
Internet context variable	Int	Internet connection active	
Location context variable	Cell	Cell ID active	
Application context variable	PM	Phone management	
	PIM	Personal information management	
	PU	Phone usage	
	CN	Connectivity	
	GM	Games	
	MM	Multimedia	
	OT	Other type	

Table 18 Context model

All context variables are synchronized with respect to time and bucketed into discrete time-slices. The duration of the time-slice determines the amount of context data available for each context reasoning step. During real-time use, the bucket size also determines how quickly an application can respond to a change in context variables. We can see a picture of the unified context signal framework in Table 19, which is defined based on the context data files captured by the ContextLog application and on the application of our previously defined context model.

Date of the week context variable							Day time-slice context variable				Phone usage and user related context variables				
Monday	**Tuesday**	**Wednes-day**	**Thursday**	**Friday**	**Saturday**	**Sunday**	**Morning (8-12)**	**Afternoon (12-18)**	**Evening (18-23)**	**Night (23 on)**	**Phone**	**Internet**	**Commu-nication**	**Applica-tions**	**Location**
1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	Total number signal is on/total instances on the time-slice				

Table 19 Context vector

The context is a vector of the vector signals that contains all the context vectors as one unified structure.

4.4.4 Capturing context data (Phase 1)

Data sources are captured through the mobile device's sensor API; in our example, they are available as context data logs from the Reality Mining project. These data are captured through an application called *ContextLog,* which runs on the mobile devices throughout the whole project.

A major challenge is to understand how to fuse the digital and real worlds. Figure 30 illustrates this challenge and depicts the conceptual architecture of the complex relationship between the environment, user and mobile device.

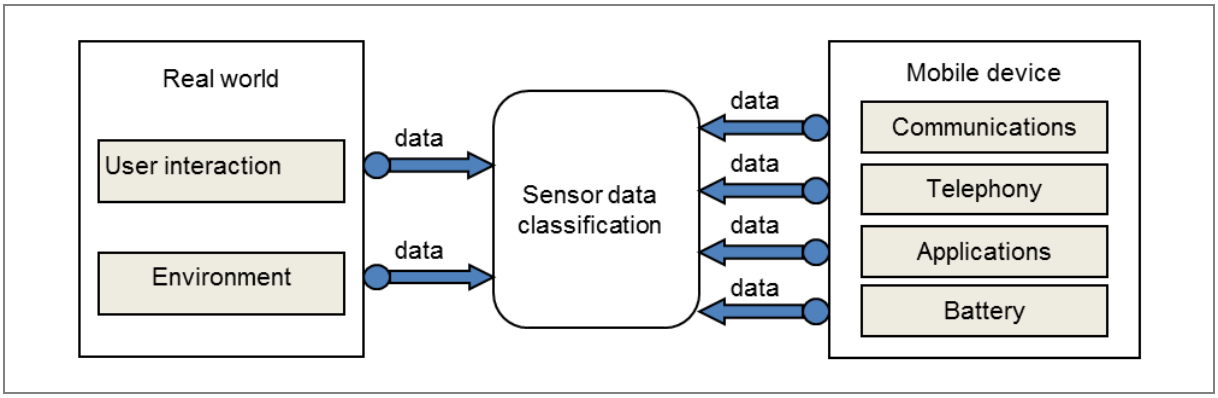

Figure 30 Conceptual architecture on real and digital world fusion through sensors

This application captures context data from the embedded sensors on the mobile devices and stores it with other variables, such as date, time, hashed user id, and data values. There are context log files for different context variables: Cell Towers, Communication, Apps, Phone On, Charge, Active, Application On.

Examples of the context group log format are shown in Figure 31, Figure 32, and Figure 33.

03-Aug-2004 15:50:19	299	-1	Voice call	Outgoing	0	201
03-Aug-2004 16:26:53	300	-1	Voice call	Outgoing	659	235
03-Aug-2004 18:42:06	301	-1	Voice call	Outgoing	0	201
03-Aug-2004 18:42:35	302	-1	Voice call	Outgoing	0	201
03-Aug-2004 18:43:05	303	-1	Voice call	Outgoing	0	201
03-Aug-2004 19:07:26	304	-1	Packet Data	Outgoing	0	NaN
03-Aug-2004 19:16:56	305	-1	Short message	Incoming	0	143
03-Aug-2004 19:17:01	306	-1	Short message	Incoming	0	143
03-Aug-2004 19:26:41	307	-1	Voice call	Outgoing	0	201
03-Aug-2004 19:43:49	308	-1	Voice call	Outgoing	48	23
03-Aug-2004 23:33:37	309	-1	Short message	Outgoing	0	201
03-Aug-2004 23:33:52	310	-1	Voice call	Outgoing	0	201

Figure 31 Reality Mining communication log sample

03-Aug-2004 01:03:34	5119.60291
03-Aug-2004 01:03:48	5119.40331
03-Aug-2004 01:04:56	5119.40312
03-Aug-2004 01:05:13	5119.40331
03-Aug-2004 01:06:16	5119.40312
03-Aug-2004 10:06:14	5123.40763
03-Aug-2004 10:10:21	5119.40792
03-Aug-2004 10:10:39	5119.40811
03-Aug-2004 10:10:51	0.00000
03-Aug-2004 10:10:51	0.00000
03-Aug-2004 10:11:44	5119.40332
03-Aug-2004 10:11:45	5119.40332

Figure 32 Reality Mining location log sample

03-Aug-2004 11:32:00	1
03-Aug-2004 11:33:02	0
03-Aug-2004 11:51:29	0
03-Aug-2004 11:53:03	1
03-Aug-2004 11:55:10	0
03-Aug-2004 11:57:32	1
03-Aug-2004 11:58:33	0
03-Aug-2004 12:26:27	1
03-Aug-2004 12:27:50	0
03-Aug-2004 12:37:54	1
03-Aug-2004 12:38:55	0
03-Aug-2004 13:03:19	1
03-Aug-2004 13:04:29	0

Figure 33 Reality Mining phone active log sample

Before using the context logs, we must first open each context log, clean them to remove noise and ensure that they have the correct format. We eliminate all null values from the files and align all dates into a unified format so that the files can be transformed into a context vector (Location, Communication, Apps, Internet and Phone usage).

This unified format implies that we must individually check each context data log to ensure that the data is of quality and all variables follow the same format: dates appear as "dd/mm/yyyy", day appears with the capitalized day name, time appears as "hh:mm:ss" and when the variable appears, a 1 is entered in the signal 'on' field. Thus, each file must be manipulated individually; however, it also ensures it can be read properly by the context modeling program.

4.4.5 Reasoning context (Phase 1)

We create a program that captures the context logs, normalizes them into smooth context signals, and, finally, blends them into a context signal vector that is used by the machine learning algorithm to infer context. The program is called "CreateVector", and it uses Equation 5 to build a context signal vector; it consists of several subprograms (program modules) that complete all necessary steps from context data acquisition up to the generation of the final context matrix:

- *"createProjectGenVars"* creates the empty context vectors (arrays) that will store the context variables
- *"obtainProjectVars"* opens the context logs files and filters the context variables
- *" fillProjectVars"* fills the context vectors with different context variables
- *"finalProjectVectorReduced"* filters unnecessary variables, creating a smooth vector
- *"finalProjectVectorDates"* formats the output file adding dates and time variables

We refer to these modules when describing how to capture context logs and transform them into a context signal vector. First, we create a context vector canvas in which we fill in the

date and time of the project and the corresponding day, and then we match the canvas with the corresponding specific day of the week (Monday through Sunday) and time frame (morning, afternoon, evening). This task is performed by a subprogram called "createProjectGenVars", which creates an empty context matrix based on information from the Reality Mining project. When initializing the context matrix, we use common start and end dates, which are "03/08/2004" and "05/05/2005", respectively, giving a total of 276 days (9,20 months) and 6624 instances. We define a time instance to determine the selection frequency of context trace variables from the log, e.g., once every hour. This module performs the necessary calculations to determine the number of instances in a day, the number of days in each month of a given project year and how to match this information with weekday and timeframe variables.

Then we apply the template to each log , this task is performed by the "obtainProjectVars", a subprogram that opens each context file log and transforms their content into context vectors (i.e., Location Vector, Application Vector, Communication Vector, Internet Vector, Phone On Vector, Phone Charge Vector, Phone Apps Vector, and Phone Active Vector), which is the output of this program.

We analyze and manipulate each context log and transform it into context vectors. Some files are straightforward to transform and require unification of context variables whose value is on or off by adding a 1 when the variable is on and a 0 when it is off; Figure 34 provides an example.

date	day	time	Mo	Tu	We	Th	Fr	Sa	Su	M	A	E	N	Ac
03/08/2004 13:23	martes, 03 de agosto de 2004	13:23:56	0	1	0	0	0	0	0	0	1	0	0	1
03/08/2004 13:24	martes, 03 de agosto de 2004	13:24:57	0	1	0	0	0	0	0	0	1	0	0	0
03/08/2004 13:38	martes, 03 de agosto de 2004	13:38:03	0	1	0	0	0	0	0	0	1	0	0	1
03/08/2004 13:39	martes, 03 de agosto de 2004	13:39:05	0	1	0	0	0	0	0	0	1	0	0	0
03/08/2004 13:58	martes, 03 de agosto de 2004	13:58:44	0	1	0	0	0	0	0	0	1	0	0	1
03/08/2004 13:59	martes, 03 de agosto de 2004	13:59:56	0	1	0	0	0	0	0	0	1	0	0	0
03/08/2004 14:03	martes, 03 de agosto de 2004	14:03:03	0	1	0	0	0	0	0	0	1	0	0	1
03/08/2004 14:04	martes, 03 de agosto de 2004	14:04:05	0	1	0	0	0	0	0	0	1	0	0	0
03/08/2004 14:04	martes, 03 de agosto de 2004	14:04:32	0	1	0	0	0	0	0	0	1	0	0	1
03/08/2004 14:05	martes, 03 de agosto de 2004	14:05:33	0	1	0	0	0	0	0	0	1	0	0	0
03/08/2004 14:17	martes, 03 de agosto de 2004	14:17:00	0	1	0	0	0	0	0	0	1	0	0	1

Figure 34 Context vector phone active

Certain context vectors are harder to create and require more extensive manipulation of the context log files to make them understandable by the model; such is the case for the application, communication, internet and location vectors.

Application vector

Applications in use are grouped by type, resulting in 6 categories: phone management, personal information management, phone usage, general management, multimedia, and other. Figure 35 provides an overview of a sample context vector.

date	day	time	Mo	Tu	We	Th	Fr	Sa	Su	M	A	E	N	PM	PIM	PU	CN	GM	MM	OT	appID
3-8-04 9:23	martes, 03 de agosto de 2	9:23:16	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	ScreenSaver
3-8-04 14:02	martes, 03 de agosto de 2	14:02:48	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	Phone
3-8-04 14:16	martes, 03 de agosto de 2	14:16:43	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	Phone
3-8-04 14:46	martes, 03 de agosto de 2	14:46:33	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	ScreenSaver
3-8-04 15:37	martes, 03 de agosto de 2	15:37:34	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	Phone

Figure 35 Context vector application usage

Communication vector

The communication log file needs to be manipulated to extract the variables and their information, communication (e.g., voice or text), direction (i.e., incoming or outgoing), whether the call was missed or answered, duration of the call and nature of the recipient/caller (whether in the phone book or not). A sample context vector is shown in Figure 36.

date	day	time	hour	Mo	Tu	We	Th	Fr	Sa	Su	M	A	E	N	text	voice	in	out	Tout	Tin	Vout	Vin	PB	Mss	Sec
02/09/2004 18:29	jueves, 02 de septiembre de 2004	18:29:38	18	0	0	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	1	0	44
02/09/2004 19:29	jueves, 02 de septiembre de 2004	19:29:50	19	0	0	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	1	0
02/09/2004 19:51	jueves, 02 de septiembre de 2004	19:51:49	19	0	0	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	1	1	0
02/09/2004 19:52	jueves, 02 de septiembre de 2004	19:52:00	19	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	1	0

Figure 36 Context vector communications

Internet

The active internet connection needs to be extracted from the communication log file because it contains the packet data information and to be converted into an on or off type. A sample vector is shown in Figure 37.

date	day	time	Mo	Tu	We	Th	Fr	Sa	Su	M	A	E	N	Internet
19/07/2004 21:40	lunes, 19 de julio de 2004	21:40:09	1	0	0	0	0	0	0	0	0	1	0	0
20/07/2004 3:38	martes, 20 de julio de 2004	3:38:42	0	1	0	0	0	0	0	0	0	0	1	0
20/07/2004 3:40	martes, 20 de julio de 2004	3:40:21	0	1	0	0	0	0	0	0	0	0	1	0
20/07/2004 3:55	martes, 20 de julio de 2004	3:55:35	0	1	0	0	0	0	0	0	0	0	1	0
20/07/2004 5:11	martes, 20 de julio de 2004	5:11:31	0	1	0	0	0	0	0	0	0	0	1	0

Figure 37 Context vector internet

Location

In this file, we create rules to sort each cell tower id into the variables we have defined: work, home and elsewhere. We use the assumptions derived from the data and behavior survey of the Reality Mining project mentioned on Chapter 3 “Section 3.3.2 Data description”. We know that students are generally at a specific location at a given time. Therefore, when capturing a cell ID, if its timestamp is in the time frame, we match it to a location and store it in a table that we use to define location. On the left side of the rule we specify the cell ID and location table values, on the right hand side the location group. For new users we can build a location table by prompting users to input their location or by

inferring it based on recurring patterns, the time of day and also identifying distinct cell towers as Home, Work, or Elsewhere according to the amount of time spent at corresponding towers at meaningful times of day.

Location derivation rules:

- *If (tower cellID in [Home cellID range])* → *Location = Home*
- *If (tower cellID in [Work cellID range])* → *Location = Work*
- *If (tower cellID in [Elsewhere cellID range])* → *Location = Elsewhere*

If we do not count with user information, we can create the cell labels by defining a set of location derivation rules based on general phone user behavior that allow us to directly map them to those locations, since rules can be pre-made, established or created by the end-user (Forstadius, Lassila et al. 2005).

We can define a set of rules to do this and explain the assumptions that support them:

- If (alarm ringing = on) or (phone charging = 1) and (time ∈ [22:00-5:00]) and (operator roaming ≠ on) → Location = Home

  Assumption: people usually charge their phone at night when they are home, the alarm is usually ringing while users are home, on both cases the user should not be roaming to a different operator that could imply the user is traveling

- If (time ∈ [08:00-20:00]) and (operator roaming ≠ on) and (movement = off) and (tilt = horizontal +30 minutes) → Location = Work

  Assumption: people usually leave their phones on their desk while working.

- If (movement = on +10 minutes)→ Location = Elsewhere

  Assumption: when people are at home or at work, they move for a limited amount of time.

A sample of the vector is shown in Figure 38.

date	day	date	hour	Mo	Tu	We	Th	Fr	Sa	Su	M	A	E	N	H	W	E	cells
25/08/2004 19:36	miércoles, 25 de agosto de 2004	19:36:18	19	0	0	1	0	0	0	0	0	0	1	0	1	0	0	2.412.700.111
25/08/2004 19:36	miércoles, 25 de agosto de 2004	19:36:42	19	0	0	1	0	0	0	0	0	0	1	0	1	0	0	2.412.702.421
25/08/2004 19:37	miércoles, 25 de agosto de 2004	19:37:42	19	0	0	1	0	0	0	0	0	0	1	0	1	0	0	2.412.700.111
25/08/2004 19:38	miércoles, 25 de agosto de 2004	19:38:10	19	0	0	1	0	0	0	0	0	0	1	0	1	0	0	2.412.702.421
25/08/2004 19:38	miércoles, 25 de agosto de 2004	19:38:23	19	0	0	1	0	0	0	0	0	0	1	0	1	0	0	2.412.700.111

Figure 38 Context vector locations

We calculate the values of context signals by dividing the number of times the signal is on by the total amount of instances of that context signal during the time frame defined. We take slices of the previously defined instance time frame for the dates on which we have data from all the context signals during the project duration.

The subprogram that takes care of this task, “fillProjectVars”, uses as an input the context matrix and each context vector created previously by the corresponding subprograms. The module opens each context vector and merges them into the context matrix, linking them by the project date and time. This task is performed by several subprograms, one per context vector; it opens each context vector and dumps its content into the context matrix. To do this task, the module determines if the context vector’s date coincides with the one in the matrix, and if so, it then calculates the percentage of total time frame of instances that each context signal is on. Figure 39 shows the resulting context matrix, which is the output of this phase, a context matrix per trace that joins and unifies all of the project data.

weekDay	Timeslice	H	W	X	text	voice	in	out	PB	Mss	PM	PIM	PU	Int	On	Ch	Ac	App
Mo	E	0.0	0.9	0.1	0.3	0.1	0.9	0.4	0.0	0.4	0.1	0.1	0.0	0.3	0	1.0	0.0	0.1
Mo	E	0.0	0.9	0.1	0.3	0.1	0.9	0.4	0.0	0.4	0.1	0.1	0.0	0.3	0	1.0	0.0	0.1
Mo	E	0.0	0.9	0.1	0.3	0.1	0.9	0.4	0.0	0.4	0.1	0.1	0.0	0.3	0	1.0	0.0	0.1
Mo	N	0.0	0.9	0.1	0.3	0.1	0.9	0.4	0.0	0.4	0.1	0.1	0.0	0.3	0	1.0	0.0	0.1
Tu	N	0.0	0.9	0.1	0.1	0.0	1.0	0.2	0.0	0.5	0.2	0.2	0.0	0.1	0	0.5	0.2	0.2
Tu	N	0.0	0.9	0.1	0.1	0.0	1.0	0.2	0.0	0.5	0.2	0.2	0.0	0.1	0	0.5	0.2	0.2
Tu	N	0.0	0.9	0.1	0.1	0.0	1.0	0.2	0.0	0.5	0.2	0.2	0.0	0.1	0	0.5	0.2	0.2
Tu	N	0.0	0.9	0.1	0.1	0.0	1.0	0.2	0.0	0.5	0.2	0.2	0.0	0.1	0	0.5	0.2	0.2
Tu	N	0.0	0.9	0.1	0.1	0.0	1.0	0.2	0.0	0.5	0.2	0.2	0.0	0.1	0	0.5	0.2	0.2
Tu	N	0.0	0.9	0.1	0.1	0.0	1.0	0.2	0.0	0.5	0.2	0.2	0.0	0.1	0	0.5	0.2	0.2
Tu	N	0.0	0.9	0.1	0.1	0.0	1.0	0.2	0.0	0.5	0.2	0.2	0.0	0.1	0	0.5	0.2	0.2
Tu	N	0.0	0.9	0.1	0.1	0.0	1.0	0.2	0.0	0.5	0.2	0.2	0.0	0.1	0	0.5	0.2	0.2
Tu	M	0.0	0.8	0.2	0.1	0.0	1.0	0.2	0.0	0.0	0.0	0.2	0.0	0.1	0	0.5	0.3	0.3
Tu	M	0.0	0.8	0.2	0.1	0.0	1.0	0.2	0.0	0.0	0.0	0.2	0.0	0.1	0	0.5	0.3	0.3
Tu	M	0.0	0.8	0.2	0.1	0.0	1.0	0.2	0.0	0.0	0.0	0.2	0.0	0.1	0	0.5	0.3	0.3
Tu	M	0.0	0.8	0.2	0.1	0.0	1.0	0.2	0.0	0.0	0.0	0.2	0.0	0.1	0	0.5	0.3	0.3
Tu	A	0.0	0.9	0.1	0.1	0.0	1.0	0.2	0.0	0.5	0.2	0.2	0.0	0.1	0	0.5	0.2	0.2

Figure 39 Sample of context matrix

The context matrix final output is created by another subprogram “finalProjectVector” that takes as input the context matrix and formats it in a way that can be fed to the machine learning algorithm for context guessing. It merges several variables into one, such is the case of: weekdays which is now one variable with values that ranges from Monday to Friday, timeframe is now one variable with values that range from Morning to Evening, location that now is one variable with three values (home, work, and elsewhere).

4.4.6 Inferring context (Phase 2)

When inferring context states, we use classifier machine learning algorithms described earlier to allow us to automatically derive a context state based on the context vector. To train these algorithms, we must define the class labels corresponding to that state. We

define a set of rules to label the classes and to train the classifiers to infer context states. Because there is no classification standard or analysis framework to do this, we propose one that will allow us to achieve a semantic understanding of context.

To label the classes, we follow the approach taken in the MavHome project (Gopalratnam and Cook 2003), and partition the user's day into zones, which correspond to activities in which the user can be involved when using a mobile device, such as making a call, using an application, and charging the phone. When defining context labels, we make a number of assumptions related to user's interaction with mobile devices (derived from the data and behavior survey of the Reality Mining project) and to the general usability of mobile device.

Assumptions related to student's lifestyle

- Students work both from their home and from their office locations.
- According to the place distribution analysis of the project, students are home mainly at nighttime for 2–6 hrs, at work for 12–15 hrs; in the other time frames, they can either be elsewhere or have no connection.
- We distinguish between days of the week (Monday through Sunday) and realize that students can work any day.
- Students use their phone primarily for personal/social use matters, i.e., in a leisure context.
- Work communication is performed primarily through email, sometimes face to face, and less often by phone.
- Personal communication is performed equally through email and face to face encounters, sometimes by phone and less often through text.
- Most phone contacts are personal contacts.

Assumptions to generic use of mobile devices usage

- Phones can be charged at work or at home.
- If users are engaged in a face to face work meeting, they are not actively using the phone.
- When users are not busy, they are more prone to play with the phone's applications (e.g., when they are commuting).

For example, we define a non-exhaustive set of rules to label our context states based on the different combinations the attribute values can take:

– `If (location[Office or Home] = 1) AND dayTimeSlice [Night] = 0) AND (Phone[Active] = 1) AND (Application[PhoneUsage] = 1) AND (communication[Out] = 1)` → **`Relaxing`**

- ((location[Home] = 1) AND (dateOfWeek[Sunday] = 0) AND (dayTimeSlice [Night] = 1)) → **Relaxing**
- If (location[Office] = 1 OR location[Home] = 1) AND (phone[Game] = 1 OR phone[Multimedia] = 1) → **Relaxing**
- If (location[Home] = 1) AND (dayTimeSlice [Night] = 1) → **Relaxing**
- If (location[Office] = 1) AND (dayTimeSlice[Night] = 0) AND (phone[Active] = 0) AND (application[PhoneUsage] = 1)AND (communication[Missed] = 1) → **Working**
- If (location[Office or Home] = 1) AND (dayTimeSlice[Night] = 0) AND (phone[Charge] = 0) AND (communication[Out] = 0) → **Working**
- If (location[Office] = 1) AND ((phone[Active] = 0 OR (Phone[Charge] = 1)) → **Working**
- If (location[Elsewhere] = 1) AND (dayTimeSlice [Morning] = 1 OR (Phone[Active] = 1) AND (application[Any] = 1) AND (connectivity[out] = 1) → **Moving**
- If (motion[Active] = 1) AND location[Elsewhere]= 1) → **Moving**
- If (motion[Active] = 1) AND internet[Active]= 1) → **Moving**

In our implementation, we use the data available in the Reality Mining project to define the labeling rules based on which context variables values are available in the logs, so in this case, the rules would be a combination of pre-set and user-defined information. The rules were constructed with both an action and a condition (Forstadius, Lassila et al. 2005) following the ontology model discussed earlier in the state of the art chapter ("Section 2.4.1 Modeling context" and Table 2), where the condition is applied to the attribute value and the action is the context state, so for example if the main attribute location of type elsewhere has a value of one then the context state is the class would be "Moving". Our classifiers will use the variables described earlier in Table 18 Context model.

In Chapter 5, we will test several classifiers with the available mobile context data to determine the most appropriate one to use in our architecture.

4.4.7 Predicting context (Phase 2)

Our user-centric approach allows us to consider a user's interaction with a mobile device. We follow a similar approach to the MavHome project (Gopalratnam and Cook 2003) to make predictions based on previously observed interactions and to manipulate representations of mobile phone interactions. For this approach, we will create symbols to refer to each zone. In our case, because our goal is to understand human behavior with mobile devices, we consider the coverage area to be user's interaction with the device throughout the user's day, and we partition it into zones that represent specific interactions. By interaction, we mean the device-related user activity, e.g., talking, using a phone application, and texting. We deploy a pure interaction-based scheme that generates an update whenever a zone boundary crossing is detected, i.e., a different interaction is detected. We assume that the context variables follow a stochastic process, and, using the same notation, we represent the interaction history of a user with a string "v_1 v_2 v_3… v_i" of symbols from the alphabet V which is the set of interactions with the mobile device, such as an application used or the cell tower it is connected to, and where v_i denotes the symbol id reported by the ith update.

We define an example of the user interaction history for one day within the zones and plot the data on a graph. We deploy a pure interaction-based scheme that generates an update whenever a zone boundary crossing is detected, i.e., a different interaction is detected. We partition the area into zones, such as application in use, location or communication. Figure 40 shows three zones based on several context variables.

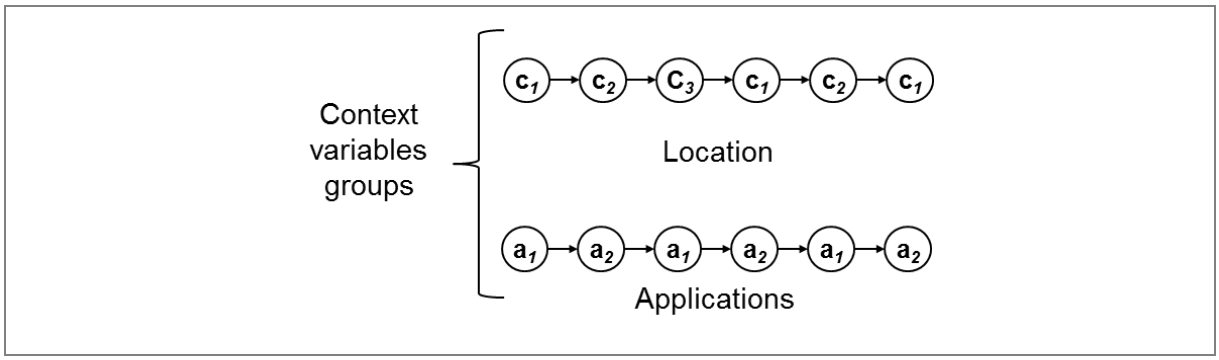

Figure 40 Mobile device usage context variables

We define the variable as symbols that are a combination of zones, changes and the value they take. We have two indexes to specify the symbol's history, one for the change between zone's and the other for the value it takes, "v_1, v_2, … v_n" where v_i denotes the zone id reported by the *i*th update.

In Table 20 we see the application usage history of what applications are being used, browser, calendar or clock.

a_1	Usage of application 1
a_2	Usage of application 2
a_3	Usage of application 3
a_2	Usage of application 2
a_{n1}	Usage of application n

Table 20 Example of application usage history

In Table 21 we can see an example of cell location history, which tower is the device connected to. We consider each cell tower to be a location

c_1	Phone connected to cell tower 1
c_2	Phone connected to cell tower 2
c_{31}	Phone connected to cell tower 3
c_2	Phone connected to cell tower 2

Table 21 Example of cell location history

The predicted context symbol will be what we use as an input to create the context signal vector to predict the context state. In Figure 41, we can see how prediction of the next symbol works for the modeled context variables, and in Chapter 6, we will test the algorithm with the available mobile context data.

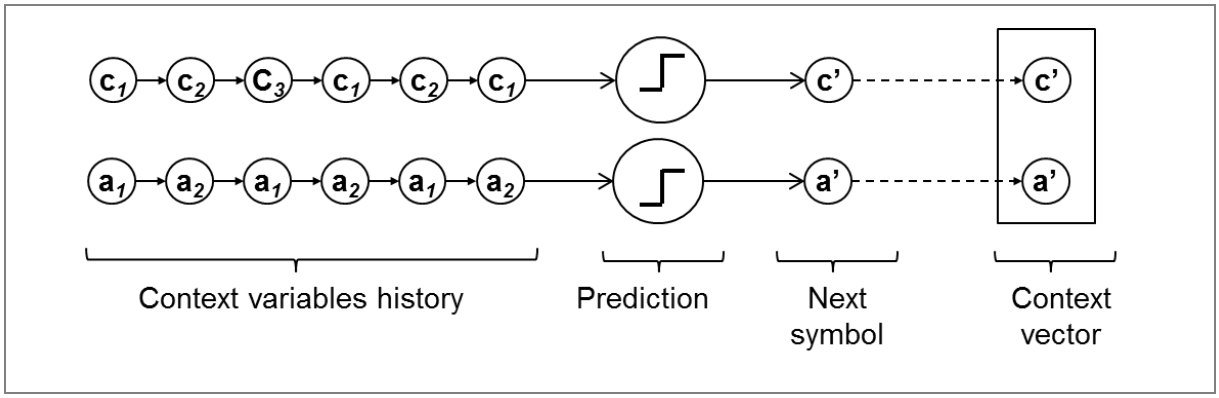

Figure 41 Next symbol prediction

The predicted next symbols will be combined to form a context vector that is used by the machine learning classifier to predict the upcoming context state.

4.4.8 Applying context (Phases 3 and 4)

Phases 3 and 4 are linked to the mobile application and its functionality; the application receives the predicted context state and uses it to adapt its results. Thus, the application logic will need to be changed on an ad-hoc basis to include the context state as an input and to contextualize the application's output, as observed in Figure 42.

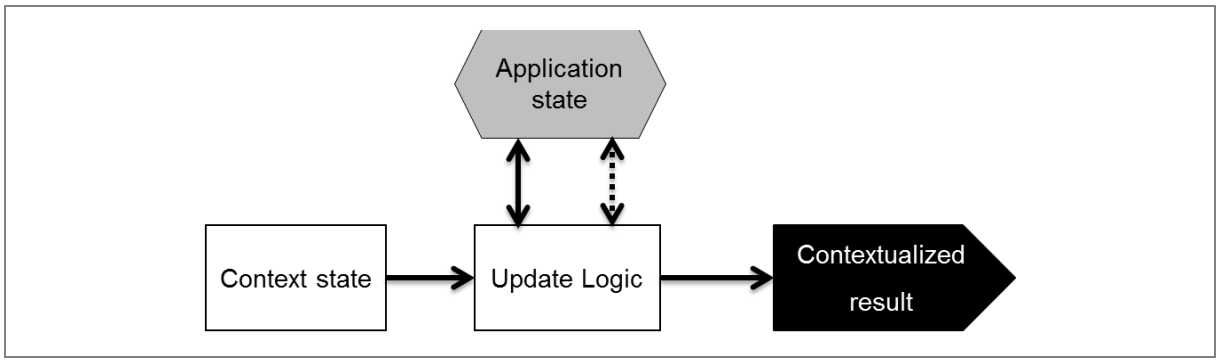

Figure 42 Applying context

In Chapter 7, we explain in-depth how applications become contextualized. We will provide a real-life example of contextualization of mobile search engines because there are several common, mobile phone-accessible variables that provide relevant information about mobile search context. Thus, by including contextual data as part of the mobile search flow, we can improve the mobile search experience.

4.5 What our architecture proposes

We apply the architecture taxonomy defined in Table 3 in Chapter 2 Section 2.5.2 and explain how our proposal adheres to it, as listed in Table 22.

	Architecture design	**Mobile oriented**	**Functionality placement**	**Reusable/ scalable**	**Signal normalization**	**Context prediction**	**Usage example**
Sorensen et al *Ad-hoc context*	Middleware	No	Device	Modular	Direct	Current	Use case
Henriscksen et al *CML*	Layer	No	Elsewhere	Modular	Post-processing	Current	Use case
Chen et al *Solar*	Middleware	No	Distributed	Modular	Direct	Current	System
Meyer et al *Context home*	Mixed	No	Distributed	Modular	Direct	Current	System
Lassila et al *Semantic*	Middleware	Yes	Distributed	Integrated	Post-processing	Current	System
Schmidt et al *Context sensing*	Layer	Yes	Device or Distributed	Modular	Post-processing	Current	Use case
Raento et al *Context Phone*	Layer	Yes	Device	Integrated	Post-processing	Current	Use case
Hofer et al *Hydrogen*	Layer	Yes	Device	Modular	Direct	Current	Use case
Fahy et al *CASS*	Middleware	Yes	Distributed	Integrated	Post-processing	Current	Use case
Chan et al MobiPads	Middleware	Yes	Distributed	Modular	Direct	Current	System
Mayrhofer et al User behavior	Layer	Yes	Distributed	Integrated	Direct	Future (partially)	Use case
✓ *Our model & architecture*	*Mixed*	*Yes*	*Device and distributed*	*Modular*	*Post-processing*	*Current and Future*	*Use case*

Table 22 Architecture taxonomy benchmark

Our proposal analyzes the existing gaps and explains how to address them,

- **Flexibility issues**: A flexible architecture with interfaces that allow for easy incorporation of new context sources in a standard manner.
  - Our proposal clearly separates the different phases to process and apply context so that the contextualization module can be reused independently of the app.
  - Our design is modular, ready to grow and scale to accommodate new contextual data independently of the sensor that captured it.
- **Mobility oriented design**: Mobility oriented design uses a light, on-line machine learning algorithm that runs smoothly on a mobile platform with limited processing power, multi-tasking, memory and battery life.
  - Our architecture uses machine learning approaches that are light and function properly in a mobile device; in our tests, we aim to maintain a relevant, historical dataset that will allow machine learning algorithms to predict context while having low runtime and memory consumption requirements.
  - Our development is independent of the mobile platform, OS or type of device (phone, netbook or laptop) and can be implemented on cross-platform programming languages, such as Python.
- **Time-frame limitations**: All of the current architectures are designed to predict context instantaneously, not at future times.
  - We add a prediction module to our architecture by using on-line predictors to forecast future context signals individually and to create a context signal vector based on the results.
  - Our classifiers predict future context states by taking as an input the future context signal vector and inferring the context states from it.
- **Contextual information processing unit:** This unit transforms and normalizes raw context to create a context signal vector that can be used independently by applications.
  - Our proposal contains a unified architecture that models context signals as a unified context vector, allow for future addition of new contextual information.
  - Our context modeling unit captures contextual information and instantly performs the reasoning of the context vector output needed to predict context states.
- **Real world data usage:** Experimental tests evaluate performance in real cases by using the mobile contextual data available at the Reality Mining project.
  - We used different types of mobile contextual data to implement our architecture and to explain the data analysis and model used.
  - We used nine months of user mobile usage contextual traces to test the machine learning algorithm performance.

4.6 Conclusions and discussion

Researchers have been demanding more context-aware approaches to mobile applications, to enable such contextualization, we have presented a generic architecture in this chapter. We defined an end-to-end framework to make mobile applications context-aware and clarified the concept of context-awareness from a mobility point of view.

We defined a mobile context-awareness architecture and explained in-depth the phases needed to capture data from sensors, to understand and extrapolate context from raw data, to predict context based on several context variables and, finally, to apply context to augment the logic of mobile applications.

We have proposed a framework that allows us to combine captured, heterogeneous sensor signals into a coherent, uniform context model that allows applications to be context-aware.

Our design has made the fundamental assumption that contextualization must occur largely outside the mobile application, as in a module. To support more powerful contextualization scenarios in the future, it will be necessary for mobile applications to become more context-aware by default. Users' activities and context information other than user location and time, such as device characteristics, will help enhance mobile apps, thereby personalizing them.

We believe that to support more powerful contextualization scenarios in the future, it will be necessary for mobile applications to become more aware of different context variables.

We explained how to capture context information using the mobile device's sensors, how to transform these data into context vectors and how to manipulate them to apply the context model and to unify their formats so that they can be understood and converted into the context matrix. When modeling contextual data, we focused on the user's interaction with the mobile device and assumed that it can be learned because it is a reflection of patterns of mobile device usage in daily life. By interaction, we mean user activity with the device, e.g., talking, using a phone application and texting.

Little of the existing context research in academia has used real-life data of mobile devices to test the performance of prediction algorithms, due to the lack of traces (Song, Kotz et al. 2006). We consider real-life data to be crucial to truly understanding how context-awareness works. In this chapter, we have used real mobile usage data from the Reality Mining project to provide an example of how to implement our proposed system, to model context and to produce a context vector that can be used by machine learning algorithms to infer context states.

We have built a standard model to use in our architecture to predict classes, given a set of attributes, a classification and a partial observation to make a statistical estimate of unobserved attribute values as the departure point for constructing new models based on the user's domain knowledge (Andreeva, Dimitrova et al. 2004).

In chapters 5 and 6, we will build a classifier based on machine learning techniques to infer context states and we also add a prediction step for it. We will define vectors based on each context variable and their possible values, explain how to label them (supporting assumptions) and determine the optimal number of traces needed. We use the Reality Mining context data to test our algorithms against real-life data by feeding our classifiers with the combined context matrix values for prediction of context states and evaluate their performance to recommend the algorithms to use in our proposal. In chapter 7 we will propose a real life example of contextualization of a mobile application.

5 Inferring context states

In this chapter, we present detailed results from context inference simulations from which we infer different classes using the mobile contextual data we have from the Reality Mining project. We test classifier algorithms across the five ten different user datasets we chose from the Project in Chapter 3, with a two-fold goal:

1. Select the classification algorithm best suited and most accurate for a mobile contextual environment.
2. Define a practical guideline on how to approach analyzing machine learning using heterogeneous data, such as the mobile data we currently have.

The datasets we use are a collection of mobile usage logs with each dataset corresponding to one user and consisting of instances, which are the attributes of the logs. In our tests, we use the WEKA (Waikato Environment for Knowledge Analysis) machine learning platform simulator (Mark Hall 2009), available at http://www.cs.waikato.ac.nz/ml/weka/, which is a comprehensive suite of Java class libraries containing many state-of-the-art machine learning and data mining algorithms and has a number of tools (classifiers) for data analysis that are used to simulate many real life environments such as in health or insurance industries to classify people (Andreeva, Dimitrova et al. 2004).

We use six different classifiers of the four types of classifier groups we have defined in the tools and background chapter:

- Rule based: Tree C4.5 and decision tables
- Bayesian family: Bayes Net and Naïve Bayes
- Instance-based algorithms: IB1
- Linear function based algorithms: support vector machines

We train these classifiers using various amounts of data to benchmark their performance and determine the number of correctly classified classes (guess rate) they achieve. We also analyze how to select the relevant feature subset, i.e., the appropriate attributes to feed into the learning algorithm. To do this, we explore what would be the optimal feature subset and its relevance to the class prediction, taking into account that the optimal features are those that give the highest prediction accuracy when used with the classifier function, and they do not necessarily have to include all the features relevant to the classification problem (Kohavi 1995).

5.1 Test framework

In machine learning, an induction algorithm is typically presented a set of training instances, where each instance is described by a vector of attribute values and a class label (Kohavi 1995). We build a classifier based on machine learning techniques to infer context states as part of phase 2 of the context-aware architecture we defined earlier in chapter 4 Figure 19 . We use a batch-type learning where all of the data are given to the learner when it starts learning (Cristianini and Shawe-Taylor 2000). We use the context vectors defined earlier in our tests to manipulate the file formats and convert them into the flat text expected by most machine learning techniques (Holmes, Donkin et al. 1994). We train the classifier by labeling a subset of the instances of the dataset to produce the model that best guesses the classes (Hsu, Chang et al. 2003).

We analyze both the learning accuracy and classification efficiency of the six algorithms and how well they perform under the constraints of mobile devices. In our analysis, we account for the number of training instances and their number of relevant and irrelevant attributes when predicting classes.

Datasets

We conducted our experiments using the 10 datasets we selected from the Reality Mining project earlier in chapter 3 "Section 3.3 Dataset description", and that have a large number of cases (average of 6000 instances) and will allow us to measure the learning/ classification efficiency. We ensure that the variables have the correct format that the classifiers require, which is homogeneous scaled to values between `[0,1]` (Hsu, Chang et al. 2003) (Bouckaert, Frank et al. 2010), which we already did in "Section 4.4.3 Modeling context data" (shown on Figure 39) when creating the context matrix.

Our samples are multi-class with three context states, working, relaxing and moving, and 17 variables: weekday, TimeSlice, Location, text, voice, in, out, PB, Mss, PM, PIM, PU, Int, On, Ch, Ac, App (as described in Figure 39 from Phase 1 of our proposed architecture in "Section 4.4.5 Reasoning context").

We use the set of rules that we described in "Section 4.4.6 Inferring context", to label the datasets into the three context states (classes) defined earlier: Relaxing, Working and Moving.

- `If (location[Office or Home] = 1) AND dayTimeSlice [Night] = 0) AND (Phone[Active] = 1) AND (Application[PhoneUsage] = 1) AND (communication[out] = 1)` → **`Relaxing`**
- `((location[Home] = 1) AND (dateOfWeek[Sunday] = 0 ) AND (dayTimeSlice [Night] = 1))` → **`Relaxing`**

- If (location[Home] = 1) AND (dayTimeSlice [Night] = 1) → **Relaxing**
- If (location[Office] = 1) AND (dayTimeSlice[Night] = 0) AND (phone[Active] = 0) AND (application[PhoneUsage] = 1)AND (communication[Missed] = 1) → **Working**
- If (location[Office] = 1) AND ((Phone[Active] = 0 OR (Phone[Charge] = 1)) → **Working**
- If (location[Elsewhere] = 1) AND (dayTimeSlice [Morning] = 1 OR (Phone[Active] = 1) AND (Application[ANY] = 1) AND (connectivity[out] = 1) → **Moving**

Table 23 shows the distributions of the datasets that we use in our classifiers.

	Variables	Name	04	08	12	20	22	23	36	53	81	93
Context state (class)	17	Working	1892	3996	2100	2730	2932	3522	3318	2536	2024	4380
		Relaxing	2188	1376	3692	3623	2946	1813	1987	1552	2100	1075
		Moving	1944	580	1051	15	688	1746	1083	1864	1336	286

Table 23 Context classes (labels) instance per user dataset

We see there are some differences in the nature of each data set that affects its labeling:

dataset 04. The class labels are well balanced with a split of almost 30% each across the entire dataset.

dataset 08. The majority of the class labels belong to the "Working" class (70%), while the remainder classes are split between "Relaxing" (20%) and "Moving" (10%) classes.

dataset 12. The class labels are split with 31% if the classes belonging to the "Working" class, 54% of the classes belonging to the "Relaxing" class, and 15% for "Moving" class.

dataset 20. Half of the class labels (c. 50%) belong to the "Working" and "Relaxing" classes with barely 3% for "Moving" across the entire dataset.

dataset 22. The class labels are split with 45% for both "Working" and "Relaxing" classes with barely 10% for "Moving" across the entire dataset.

dataset 23. Half of the class labels (c.50%) belong to the "Working" class while the remainder are split almost evenly, c.26% for "Relaxing" and c.25% for "Moving" across the entire dataset.

dataset 36. Half of the class labels (c.52%) belong to the "Working" class while the remainder are split between the "Relaxing" (c.31%) and a smaller amount of c.17% for "Moving" class.

dataset 53. The class labels are evenly split with approximately 40% for "Working", 25% for "Relaxing" and 30% for "Moving" across the entire dataset

dataset 81. The class labels are well balanced with a split of c.37% for the "Working" class, c.38% for the "Relaxing" class and c.24% for the "Moving" class.

dataset 93. The majority of the class labels belong to the "Working" class (76%), while the remainder classes are split between "Relaxing" (19%) and barely 5% for "Moving" class.

We specify the class distribution because it affects the performance of the classifiers, since some of them are prone to work better with two rather than three classes.

Test description

We ran a large batch of classification tests to infer the context states and report their results:

- Training the classifier: We test the classifiers by gradually increasing the training size of the instances chosen of the dataset and test over the remaining ones. To do this, we change the training size parameter ("percentage split") available in (Bouckaert, Frank et al. 2010) WEKA from 10% to 90% incrementing it by 10.
- Adding noise: We repeat the training tests adding incremental noise to our datasets using WEKA's "add noise" option that inserts by default 10% of noise into the dataset, introducing noise data to a random subsample of the dataset (Bouckaert, Frank et al. 2010).
- Changing behavior patterns**:** We include a disruption by combining two different datasets to create a new one that changes the usage pattern.
- Runtime: We measure the algorithm average CPU seconds to build the model, this information is crucial since it will determine if we can install the full functionality of the classifier in the mobile device or if we will need to make use of an external source when creating the model, addressing the restrictions of mobile devices described in "Section 3.1.4 Challenges".
- Attribute analysis: We repeat our accuracy tests variable groups and analyze each variable's relevance of the dataset individually.

We present a comparison of the performance of the algorithms and evaluate the impact of changing the values of the training set size, noise level and number of attributes on the results. In all our tests, we randomize the input order and the training/test split to get a more accurate picture of the classifier performance and to be able to generalize beyond the examples in the training set (Domingos 2012). We therefore repeat our tests 10 times changing the random seed and calculate the confidence interval using a confidence level of 95%.

Performance metrics

We determine the algorithm **accuracy**, which is its guess rate calculated as the percentage of classes properly identified within the dataset.

We also determine the **F Score** (also called F1 Score) value shown in Equation 6, which measures the relation between precision and recall, as defined by (Bouckaert, Frank et al. 2010) and (Powers 2011) with the purpose of understanding the type of mistakes the classifier makes.

$$F\ Score = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Equation 6 F Score formula

Where:

- **Precision** (or confidence) denotes the proportion of predicted positive cases that are correctly real positives, i.e. the proportion of instances that are correctly classified as belonging to one class among all the instances classified as belonging to that class.
- **Recall** denotes the proportion of real positive cases that are correctly predicted positive, i.e. the proportion of instances classified as belonging to one class among all the instances that belong to that class, this value indicates how much of the class was captured when classifying.

We illustrate how to calculate precision, recall and F Score with an example extracted from our classification tests results using WEKA's "confusion matrix" which shows how many instances have been assigned to each class captured. The row shows the classes and the columns the predicted elements. The number of correctly classified instances is the sum of diagonals in the matrix, the others values are the all others are incorrectly classified classes (Bouckaert, Frank et al. 2010).

Figure 43 shows an output of a confusion matrix when classifying a dataset.

```
=== Confusion Matrix ===

   a   b   c   <-- classified as
 242  41  33 |   a = Relaxing
  32 413  46 |   b = Working
  37  41 305 |   c = Moving
```

Figure 43 Example of WEKA's confusion matrix

The number of correctly classified instances would be 960 (242 of class a, 413 of class b and 305 of class c) and the incorrect ones would be distributed as follows:

- 41 instances of class a get misclassified as `class b` and 33 as `class c`
- 32 instances of class b get misclassified as `class a` and 46 as `class c`

- 37 instances of class c get misclassified as `class a` and 41 as `class b`

We calculate the precision and recall values when classifying class a and show the result in Figure 44

$$precision = \frac{242}{242 + 32 + 37} = 0.78\,, recall = \frac{242}{242 + 41 + 33} = 0.77\; F1\; score = 2 \cdot \frac{0.77 \cdot 0.88}{0.77 + 0.88}$$

Figure 44 Calculating Precision, Recall and F1 Score for class = a

The overall `precision`, `recall` and `F1 score` are calculated as weighted averages of the values obtained for each class.

We use all these measurements when evaluating the classifiers performance to allow us to choose the one that has the best balance between these metrics.

Programming tools

We ran our programs on a Dell Latitude 4200 laptop with Windows 7 OS. We use the 3rd edition of WEKA open-source simulator software, issued under the GNU General Public License, to run our classifiers (Mark Hall 2009). It is an environment that allows access to a variety of machine learning techniques, with both pre- and post-processing functionalities, and contains a "collection of visualization tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces" (Holmes, Donkin et al. 1994). WEKA provides an integrated environment in which we can access a variety of machine learning techniques through an interactive interface and provides analysis tools that allow us to understand the results of our tests more in depth and has been chosen as one of the favorites for data mining tests (Andreeva, Dimitrova et al. 2004). WEKA allows us to upload our context vectors, using the .csv format, and choose the attributes to test as well as the number of instances to use when training the classifier.

5.2 Context states inference results

We analyze and report the results from an empirical evaluation of six classifiers described in our tools and background chapter and simulated using the WEKA test environment, also described earlier. Our goal is to perform in-depth and thorough testing of these algorithms over different datasets to understand how they work and determine their performance when classifying real-life mobile data. We analyze the results of each algorithm on all datasets for accuracy when classifying, the type of errors they make (F Score value), their runtime (time to create the model) and their resistance to noise.

We ran the six algorithms on each dataset, increasing the training size from 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, and 90% of the data sets' instances, as we can see in Table 24, and test them over the remaining ones. We randomize the input order and the

training/test" and repeat our tests 10 times changing the random seed calculating the confidence interval using a confidence level of 95% as explained in "Section 5.1 Test framework".

Dataset training percentage	Number of instances per dataset									
	4	8	12	20	22	23	36	53	81	93
10%	602	595	684	637	657	708	639	595	546	574
20%	1205	1190	1369	1274	1313	1416	1278	1190	1092	1148
30%	1807	1786	2053	1910	1970	2124	1916	1786	1638	1722
40%	2410	2381	2737	2547	2627	2832	2555	2381	2184	2296
50%	3012	2976	3422	3184	3284	3541	3194	2976	2730	2871
60%	3614	3571	4106	3821	3940	4249	3833	3571	3276	3445
70%	4217	4166	4790	4458	4597	4957	4472	4166	3822	4019
80%	4819	4762	5474	5094	5254	5665	5110	4762	4368	4593
90%	5422	5357	6159	5731	5910	6373	5749	5357	4914	5167

Table 24 Datasets instances used to train the classifiers

5.2.1 Classifiers results without noise

We test the accuracy of the classifiers first without noise to determine their tolerance to it and their robustness.

The graphs in Figure 45, Figure 46, Figure 47, Figure 48, Figure 49, and Figure 50 report the classifying accuracy of each algorithm over all the datasets.

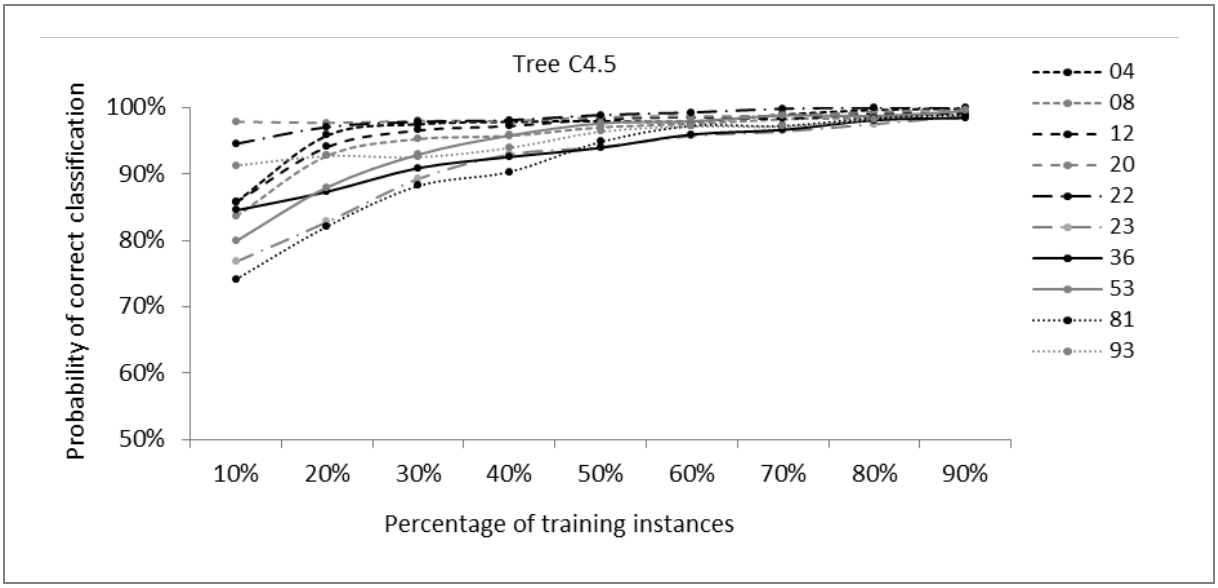

Figure 45 Tree C4.5 classifier context state guess rate

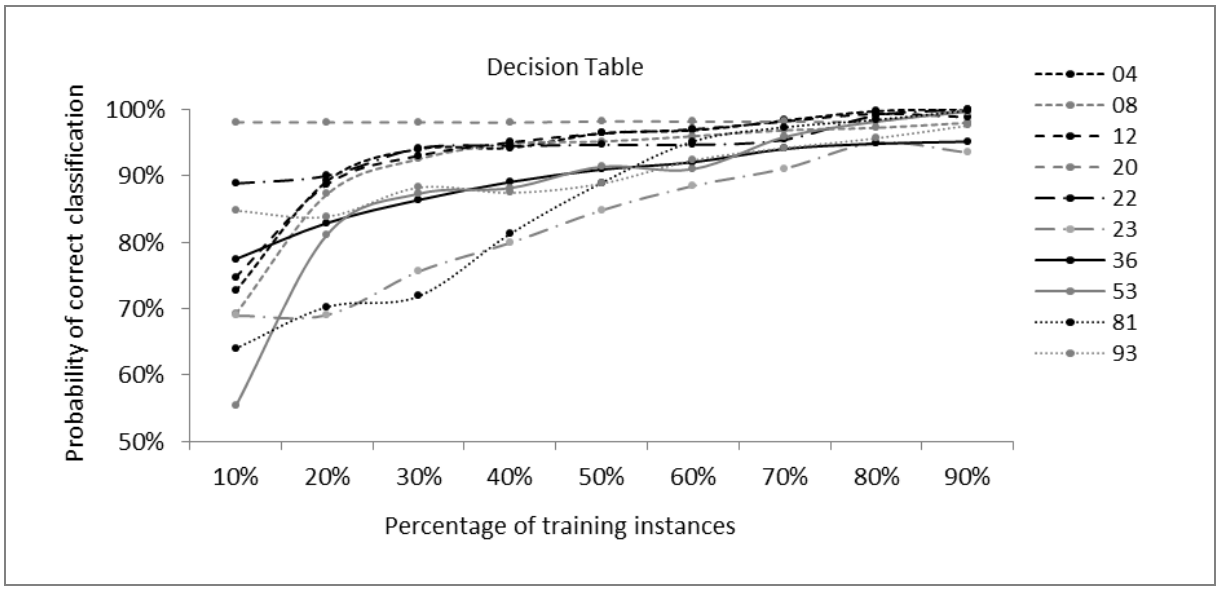

Figure 46 Decision Table classifier context state guess rate

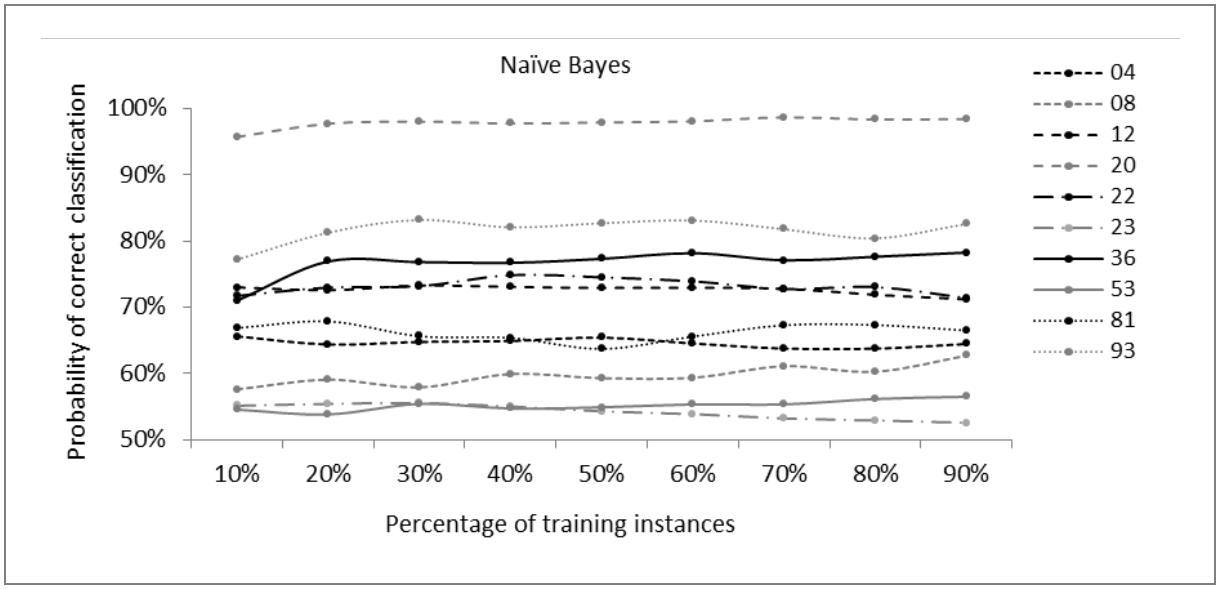

Figure 47 Naïve Bayes classifier context state guess rate

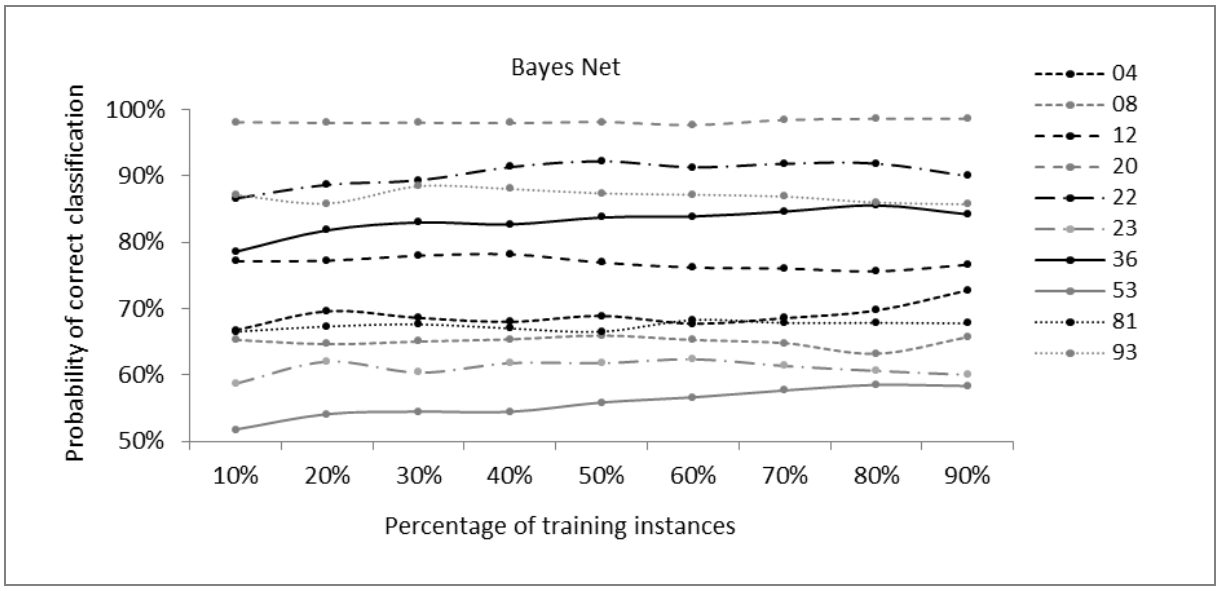

Figure 48 Bayes Net classifier context state guess rate

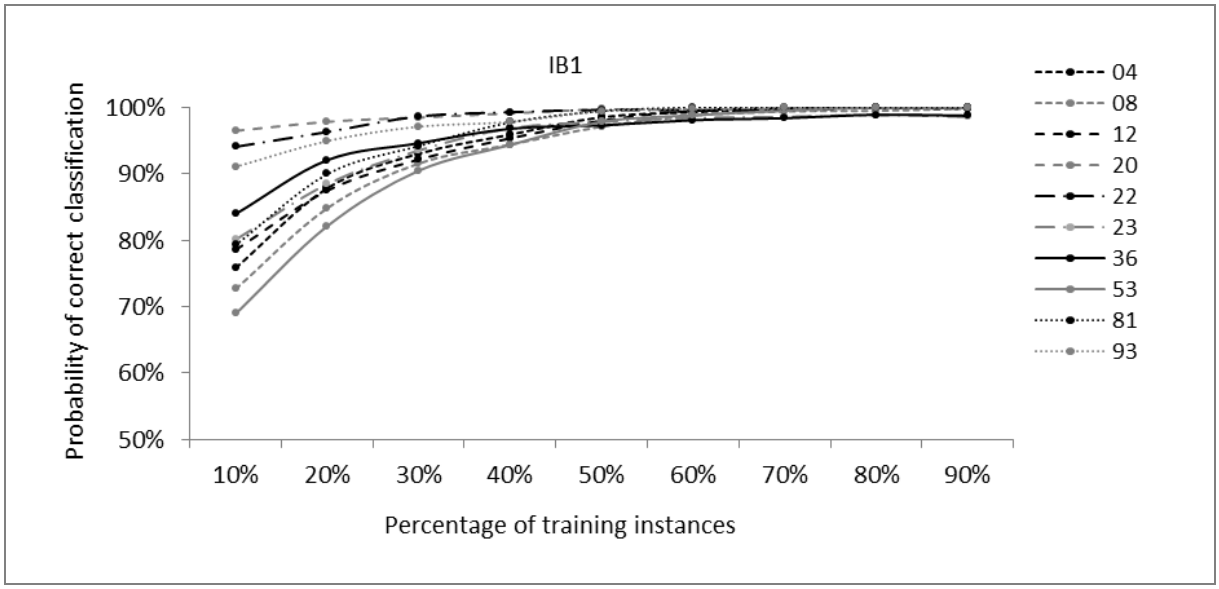

Figure 49 IB1 classifier context state guess rate

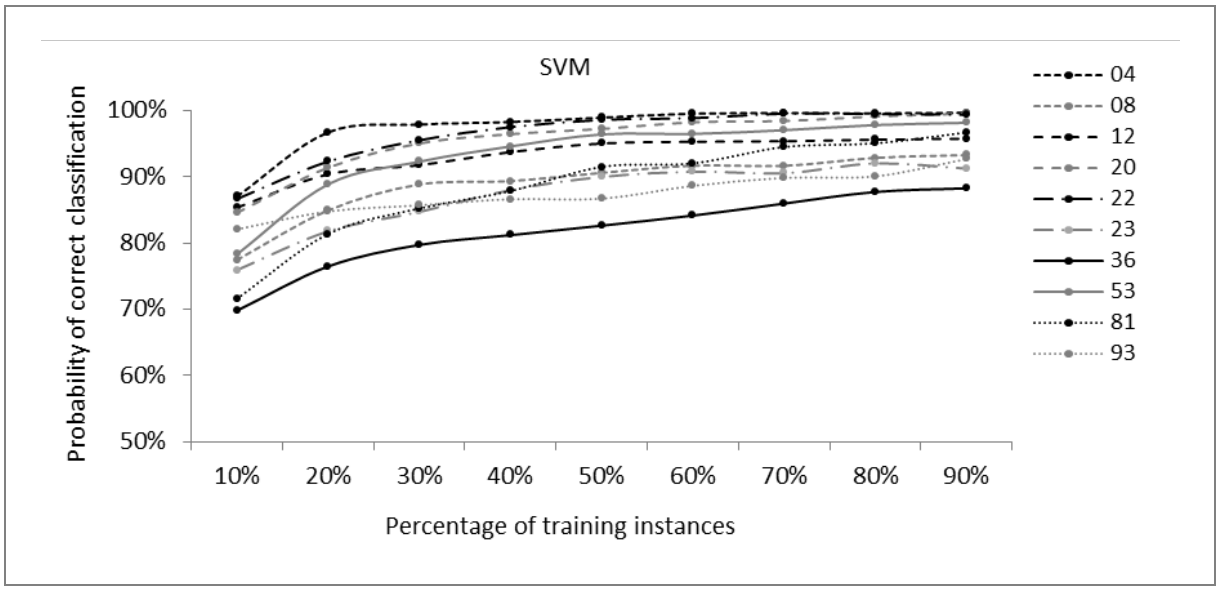

Figure 50 SVM classifier context state guess rate

The graphs in Figure 51, Figure 52, Figure 53, Figure 54, Figure 55, and Figure 56 report the F Score results of each algorithm over all datasets.

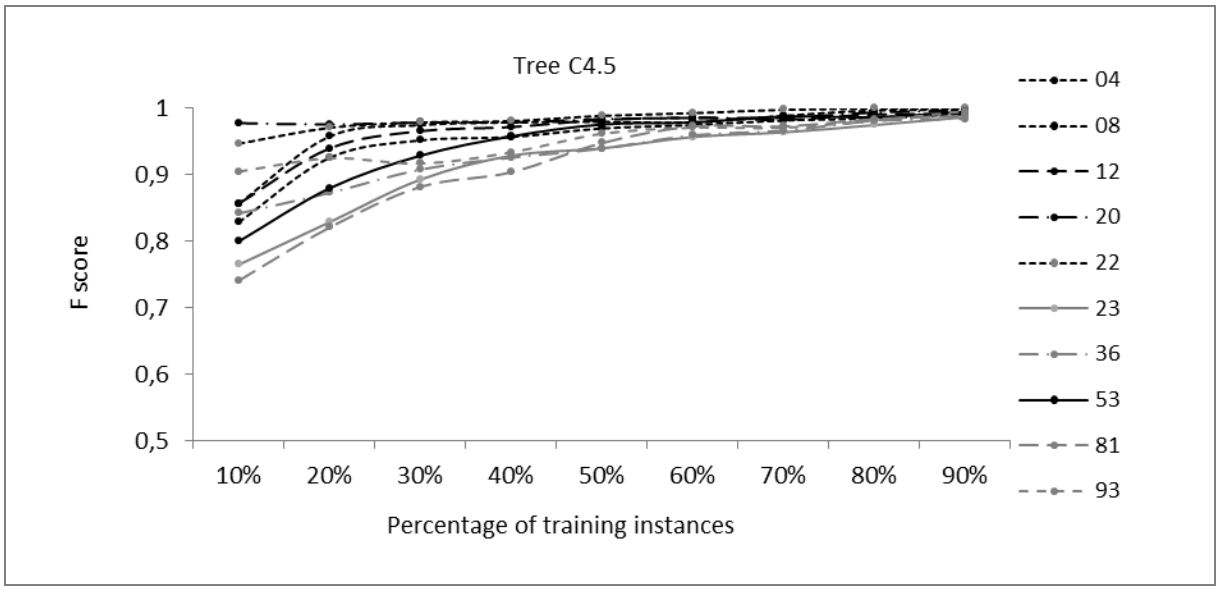

Figure 51 Tree C4.5 classifier F Score result

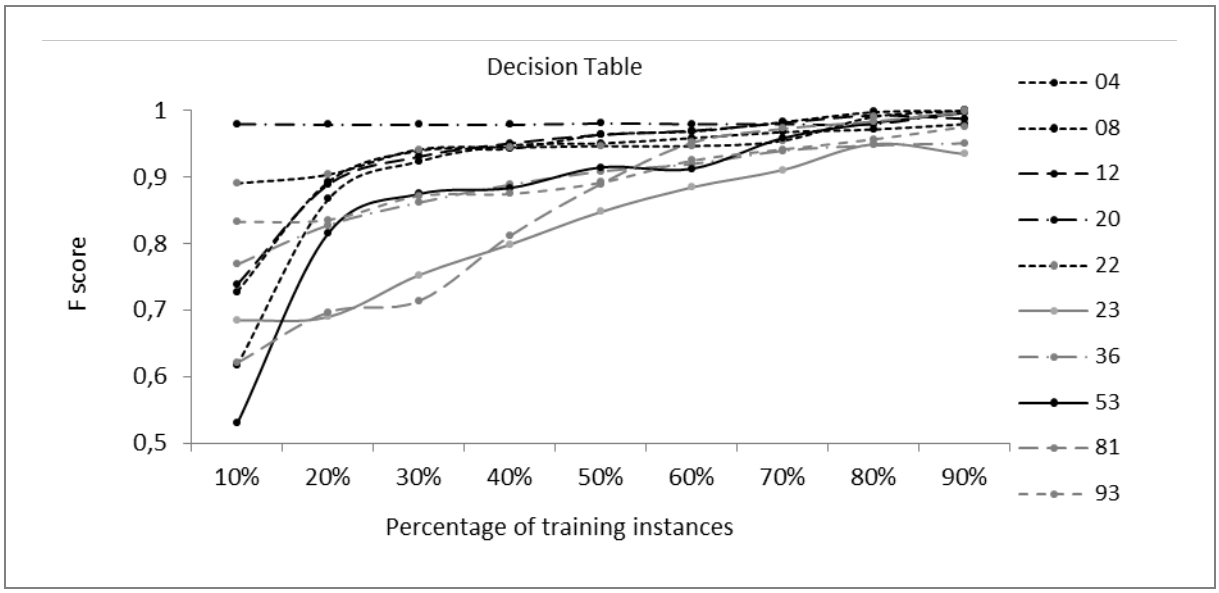

Figure 52 Decision Table classifier F Score result

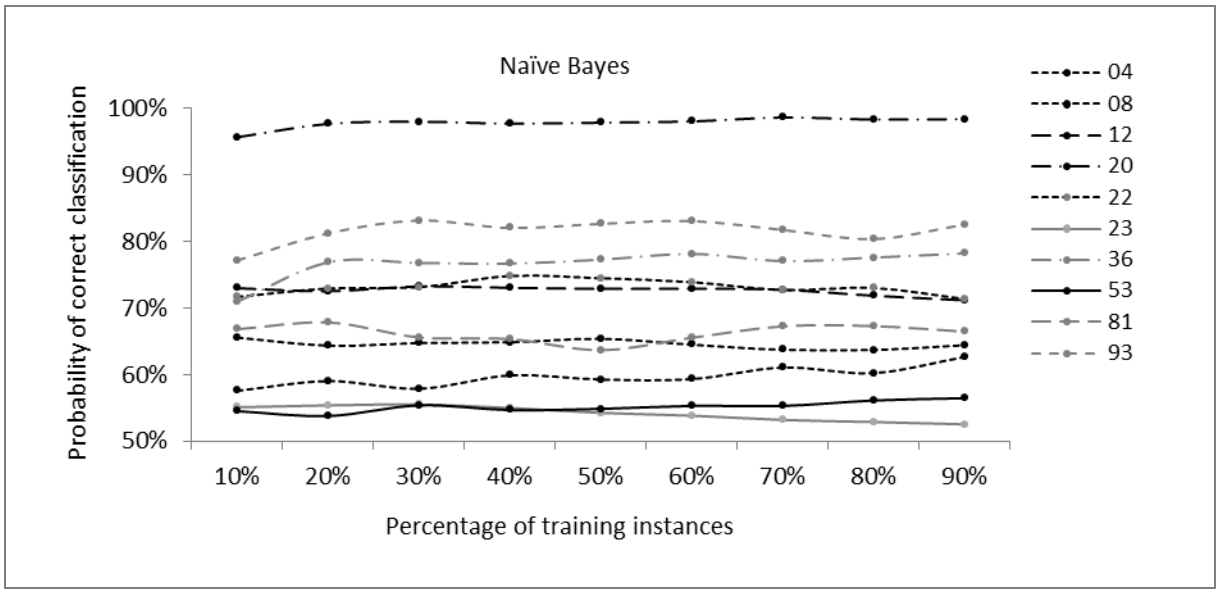

Figure 53 Naïve Bayes F Score result

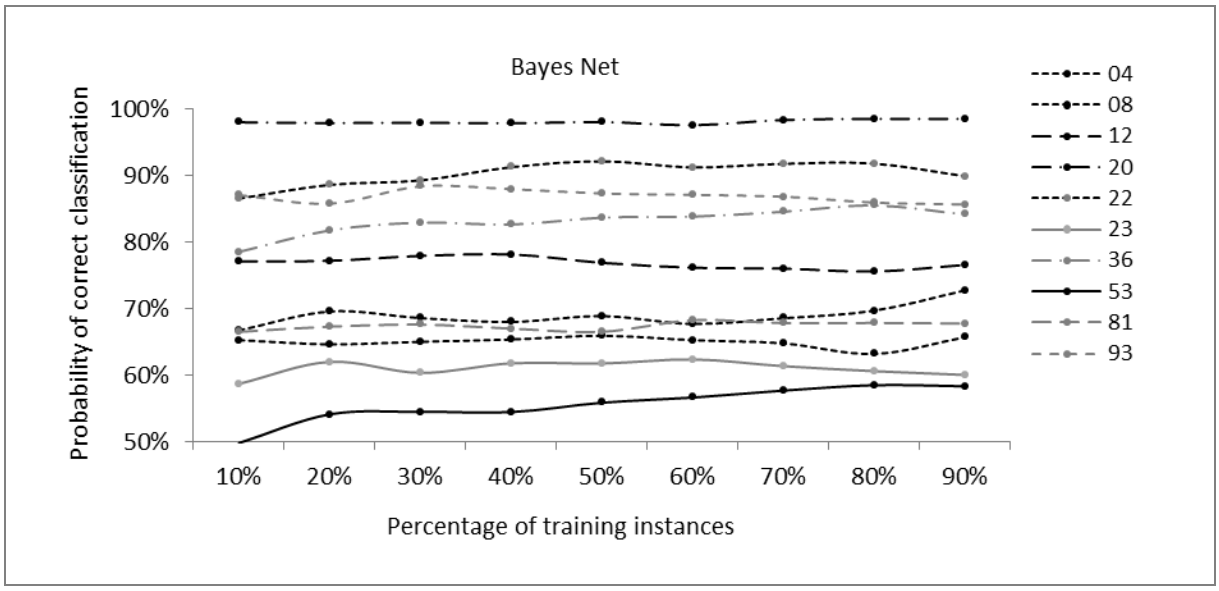

Figure 54 Bayes Net classifier F Score result

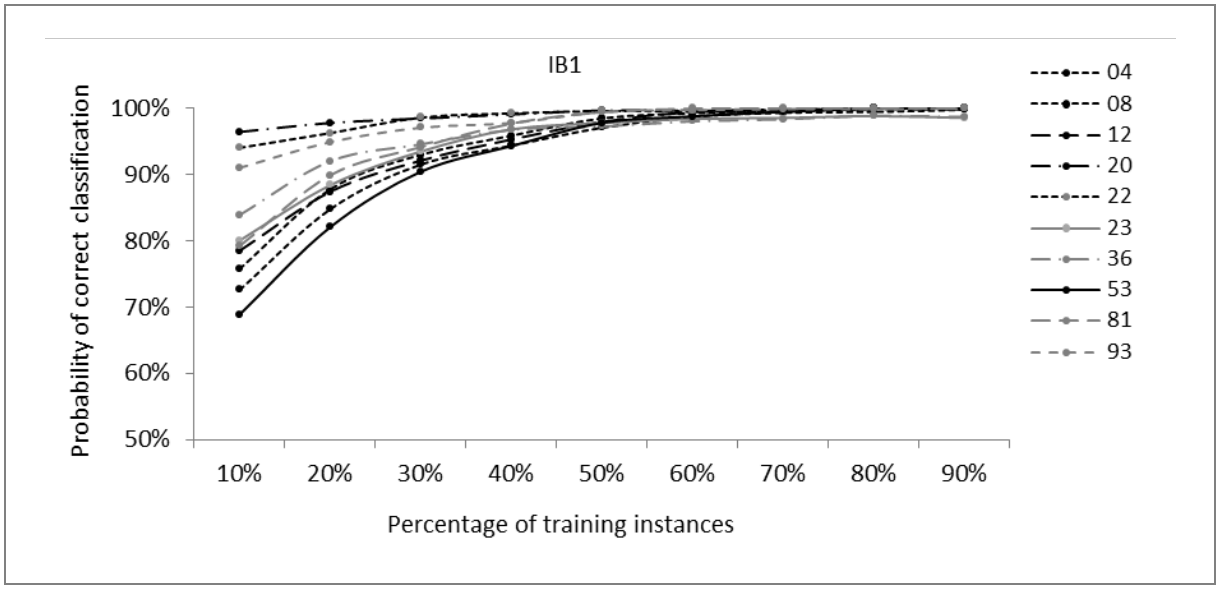

Figure 55 IB1 classifier F Score result

Figure 56 SVM classifier F Score result

The results from the tests show that for all classifiers the **average accuracy rate** is 86% in all user datasets except for the Bayes family. The best performing classifiers are the IB1 and Tree C4.5 with an average result of c.95%, the Decision Table and SVM with an average result of c. 91% and last the Bayes family (Naïve Bayes and Bayes Net) with an average of 70% to 75% respectively. The average variance and confidence interval are both below 2%.

The case of the Bayesian family classifiers (Naïve Bayes and Bayes Net) is different; their overall classification accuracy is in average 70% and 75% respectively, and it is not affected by the changing of the training size, they both have an average standard deviation of 0.13 in the accuracy results when increasing the training size and of 0.073 in the difference in the accuracy result when increasing the training size. Their accuracy results fluctuate from 65% to 84% in average for both classifiers. This is due to how the algorithm builds the model when it is being trained, if a class and a given attribute value never occurs together, the training set wipes out the information about it (Kotsiantis 2007), in our mobile usage scenario this situation happens very often regardless of the number of instances. So for example, it might happen on the training test instances the attribute `internet` is not used, and therefore it would get wiped out from the model by the classifier, but in fact this attribute is relevant since when it is activated, it indicates that user is on the move (`context state moving`) as we defined in "Section 4.4.6 Inferring context".

The **user** on which the classifiers perform the best in average is user 20, with a result of c. 98%, and the user on which the classifiers perform the worst in average is user 53 with a result of c.55%. On the rule-based algorithm the user that shows slighter worse results is user 23.

We analyze the datasets of these users to find out the relation with the accuracy results found,

- In the Reality Mining project survey ("Section 3.3.2 Data Description") user 20 describes itself as having a very regular pattern, heavily using the phone both for work and personal purposes calling and sending messages. Its dataset has half of its instances labeled for as "Working" and the remainder for "Relaxing" and "Moving". When we analyze the dataset, we confirm that there is a constant use of the phone for calls with only 2% of the vectors having a null value on the call/text variable, in the first instances. The combination of the high amount of phone usage and pattern repetition allows the classifiers to better classify into contexts (classes) the instances.
- On the other hand, user 53 describes itself as having a somewhat regular pattern, using the phone both for work and personal purposes mainly sending texts and making calls, while prioritizing face to face meetings for personal communication. Its dataset has 40% of its instances labeled for as "Working" and the remainder for "Relaxing" and "Moving". When we analyze the dataset, we confirm that there is a constant use of the phone, and that only 5% of the time the phone is inactive, which happens scattered throughout the instances, in all time frames and week days. This lack of pattern in the phone activity and regular pattern usage is what causes the classifiers to perform worse when classifying the instances into contexts (classes).
- Despite the fact that user 23 describes itself as very predictable and to use heavily the phone both for work and personal uses, the accuracy is worse because the dataset has a lot of inherent noise on it. When we analyzed it in detail, we discover many blank values in the location and phone active attributes while the phone is on, when they should appear as filled, which complicates the learning of the usage rules.

The average **percentage of training** instances needed to reach an average accuracy above 90% is c.20% of the total dataset. We will use this training size in the rest of our tests. When the training size reaches 50% of the dataset, the accuracy improvement rate slows down below 1% in average. Moreover, in some cases, there is an over-fitting effect on which the accuracy levels reach 100% in some users, when the train size gets larger (60%, 70%, 80%, or 90%). The classifier is adapting itself too much to the training set and therefore it will not work well when different datasets appear. The explanation for this is in the phone usage patterns of the users and the labeling rules. So for example, some users have a balanced distribution of classes and an almost direct mapping between location and phone usage, i.e. on each location, "Home", "Work", and "Elsewhere", there are similar patterns in phone usage. This is the case of user 04 that has an evenly split of location context variables and uses the phone mainly for internet and to make outgoing communication (voice and text) while being elsewhere. This implies that if these classifiers were to be used, the training size cannot exceed a threshold of 30% or 40% of the dataset.

In all cases, the **F Score** results are consistent with the accuracy rates of the classifiers, with an average of 0.86 overall, an average of c.0.7-0.75 for the Bayes family classifiers and of c.0.9-0.95 for the rest of the classifiers (the same happens with the average variance and

confidence interval, which are both below 2%). The high result close to one indicates that both precision and recall values are high: therefore the classifiers are correctly capturing the majority of the classes with very few instances misclassified as to belong to once class when in fact they belong to another (less than 15% in average).

5.2.2 Classifiers results adding noise

We repeat the tests adding incremental noise to the datasets to test how well they tolerate it and to analyze how much it affects its performance, since noise in data means that there will be errors in classification.

The graphs in Figure 57, Figure 58, Figure 59, Figure 60, Figure 61, and Figure 62 report the classifying accuracy of each algorithm over the datasets adding noise.

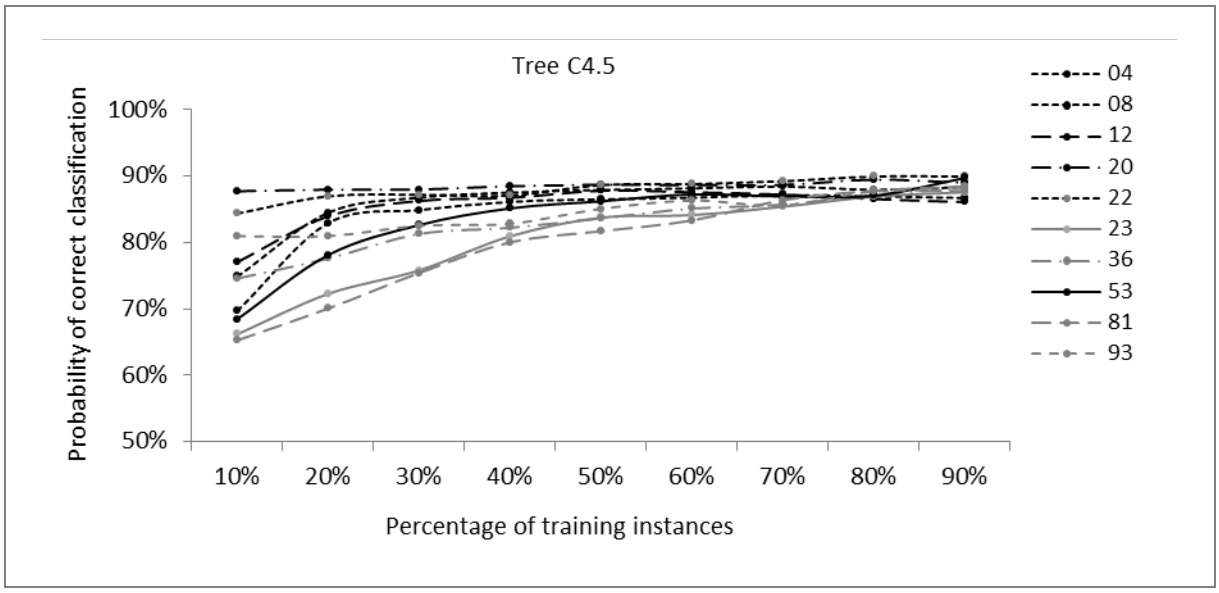

Figure 57 Tree C4.5 classifier context state average guess rate with noise

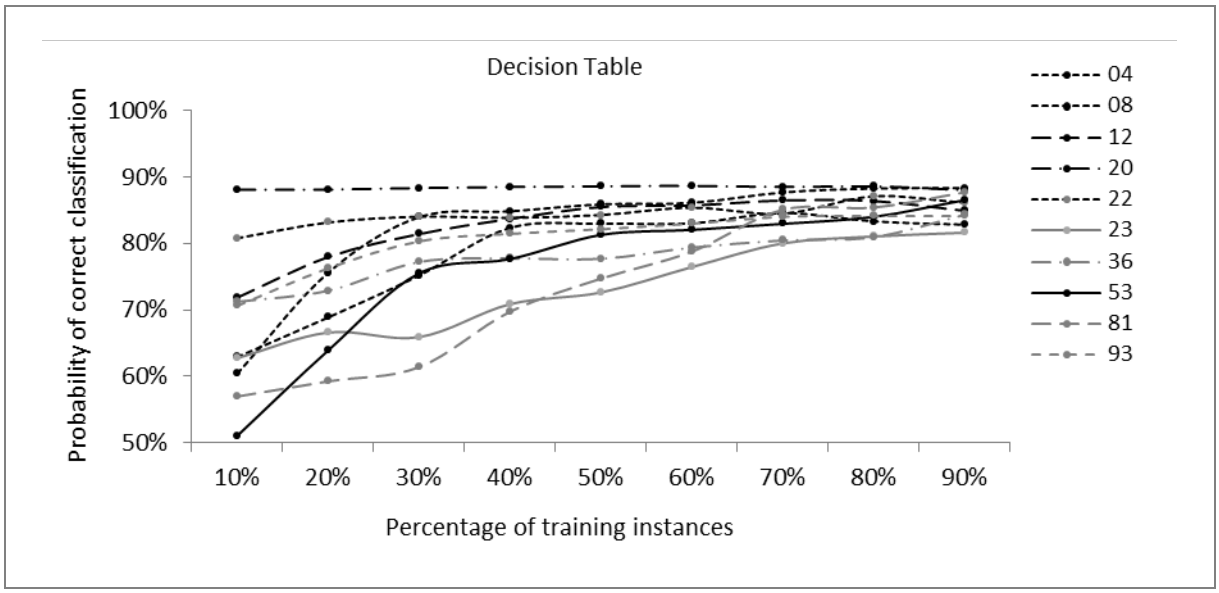

Figure 58 Decision Table classifier context state average guess rate with noise

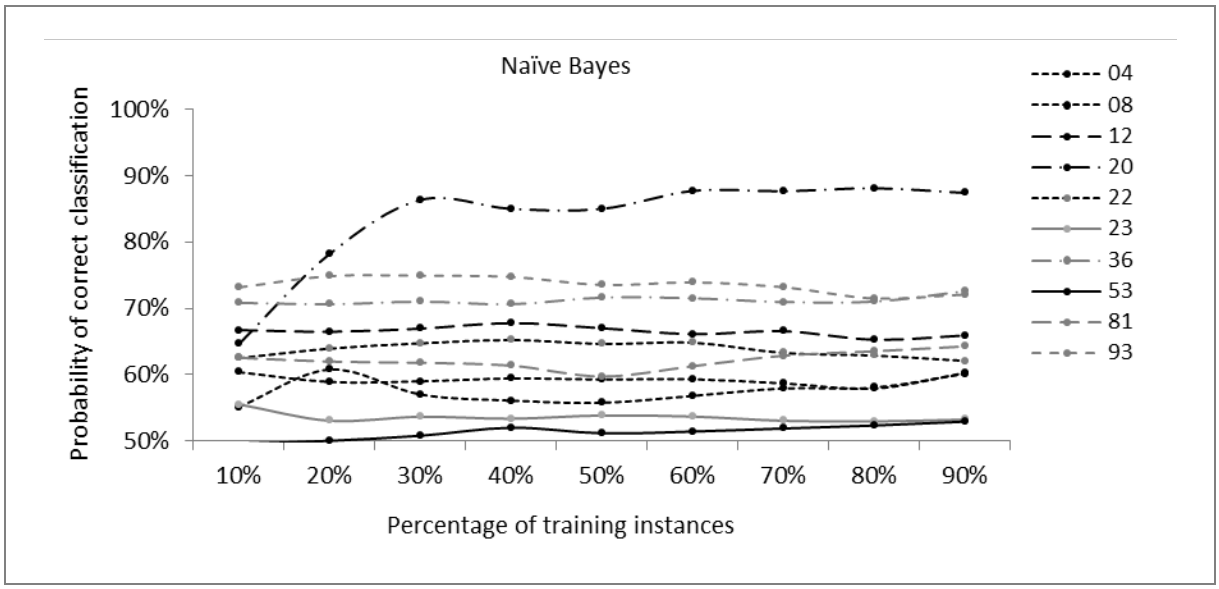

Figure 59 Naïve Bayes classifier context state average guess rate with noise

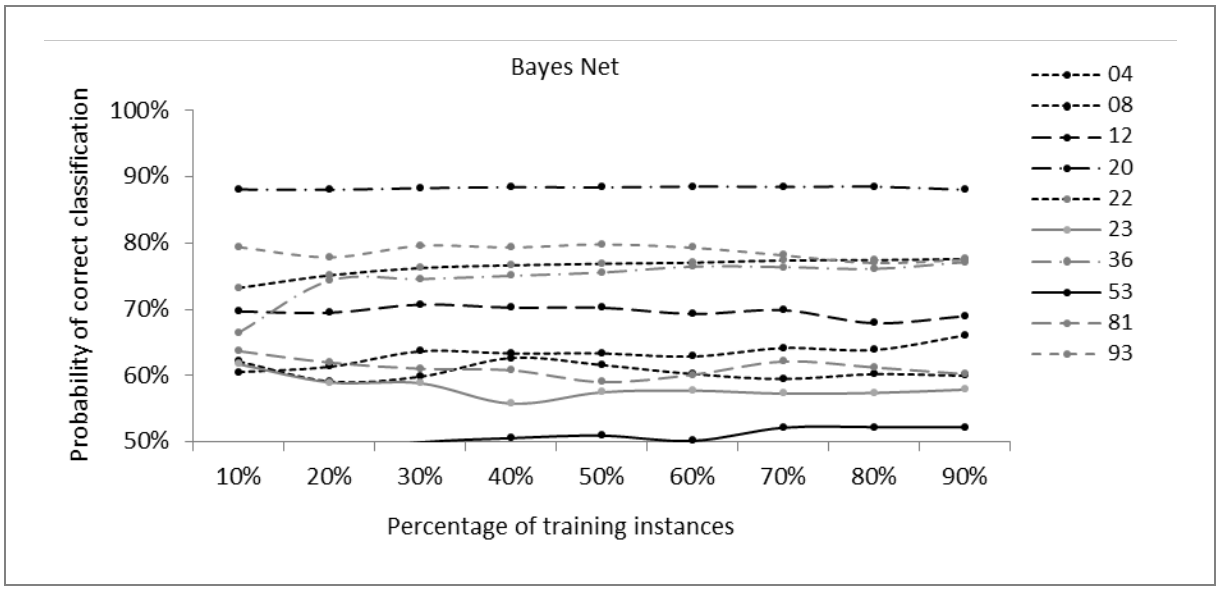

Figure 60 Bayes Net classifier context state average guess rate with noise

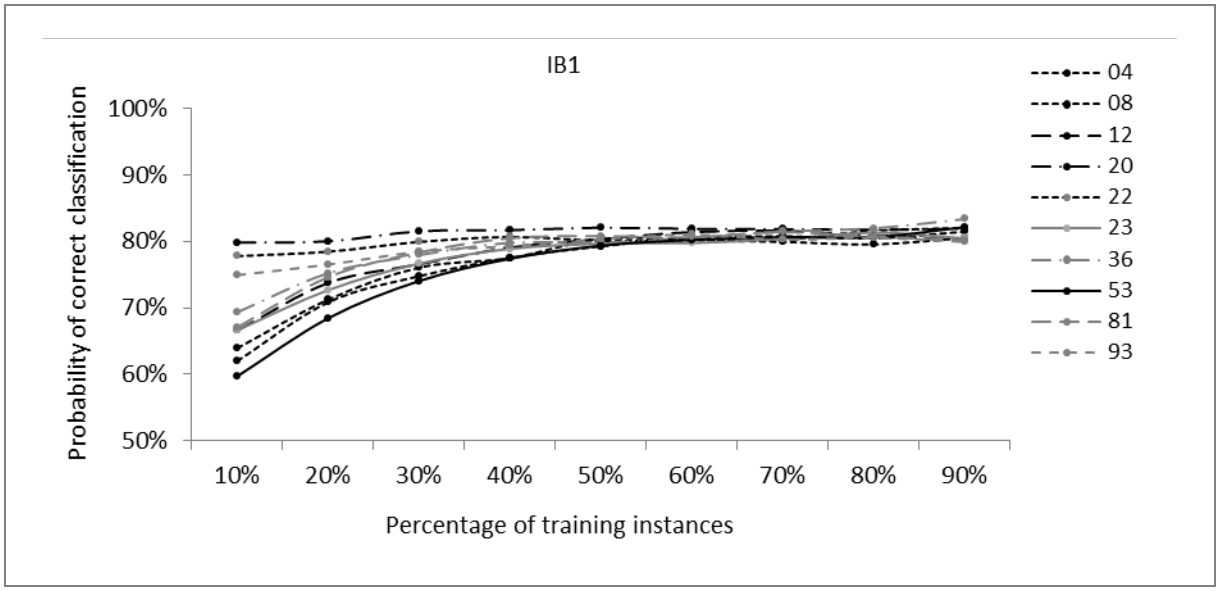

Figure 61 IB1 classifier context state average guess rate with noise

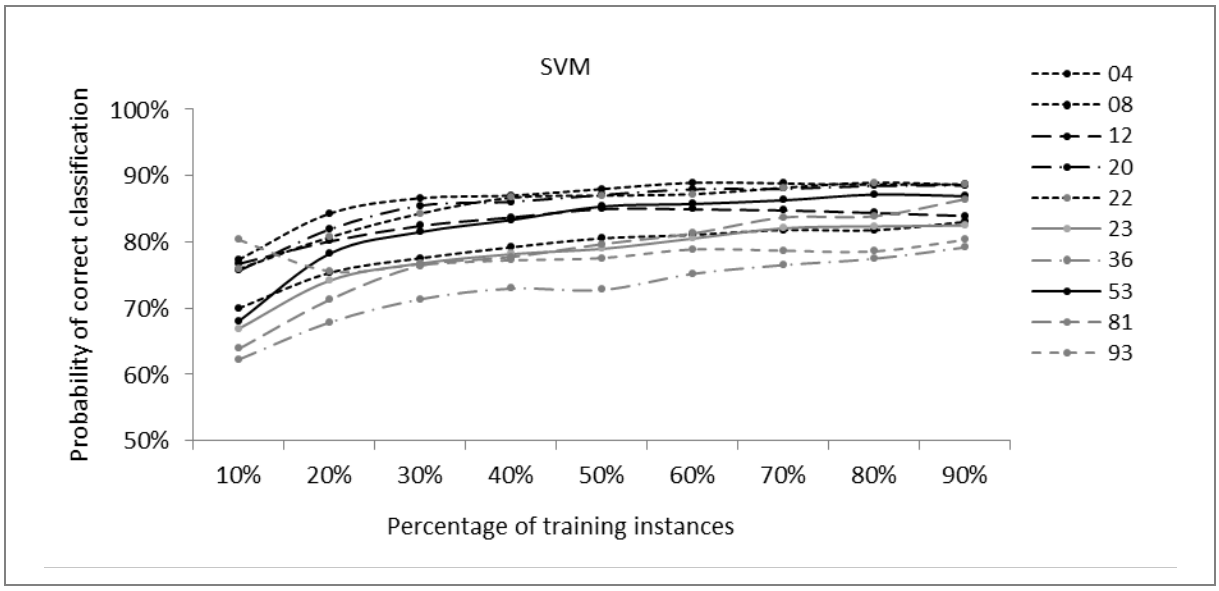

Figure 62 SVM classifier context state average guess rate with noise

The graphs in Figure 63, Figure 64 , Figure 65, Figure 66and Figure 67 report the results for the F Score when adding noise.

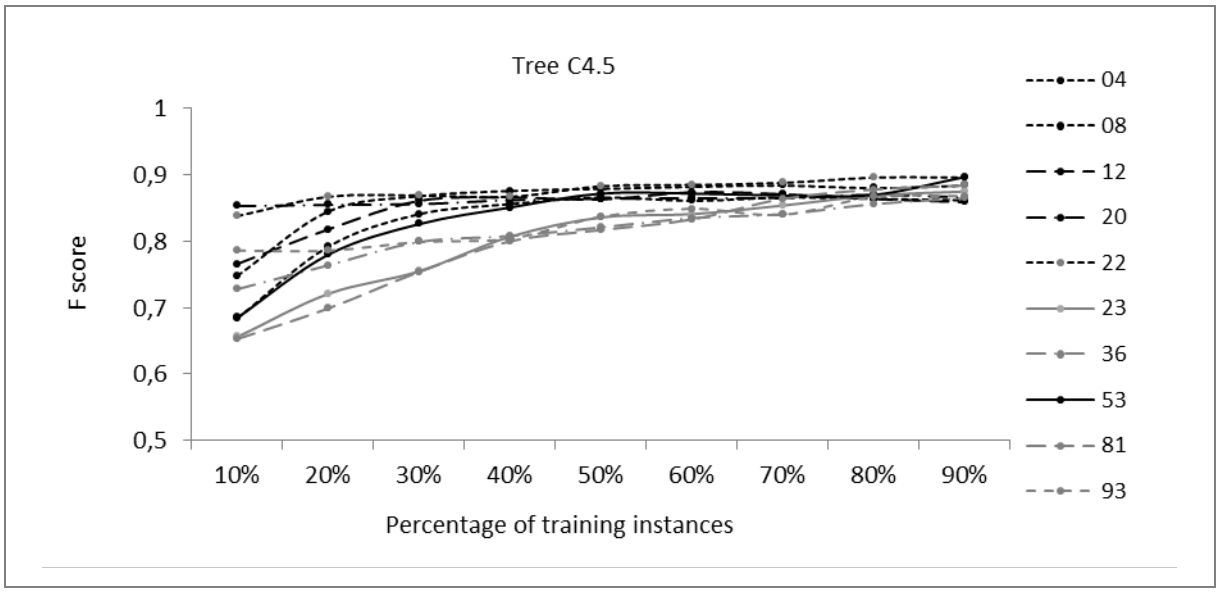

Figure 63 Tree C4.5 classifier F Score result with noise

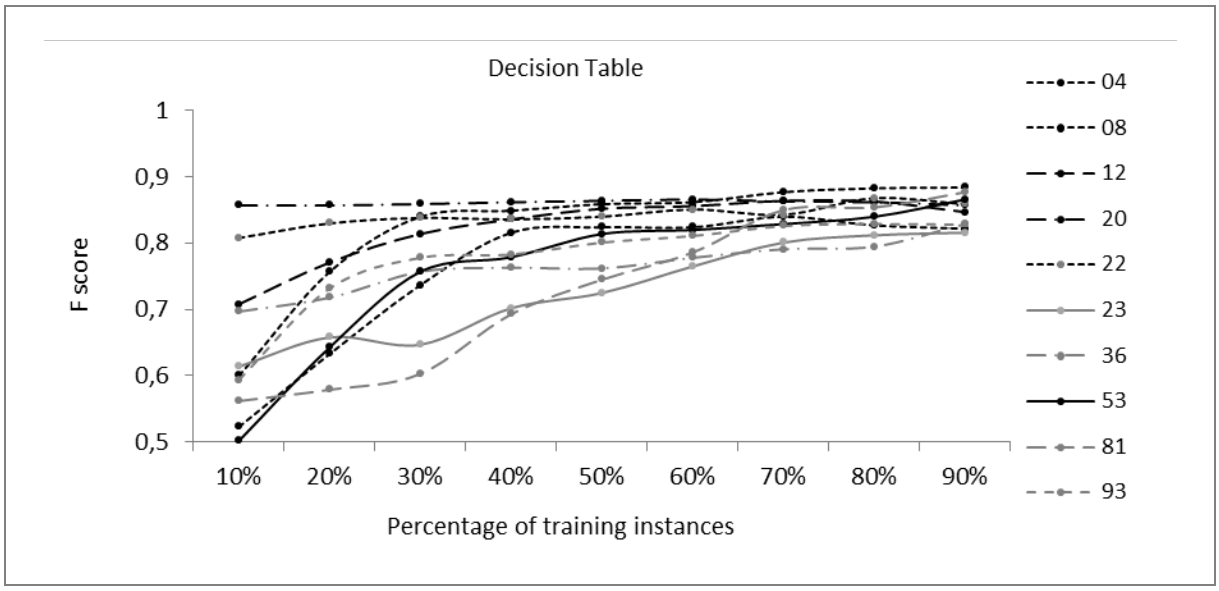

Figure 64 Decision Table classifier F Score result with noise

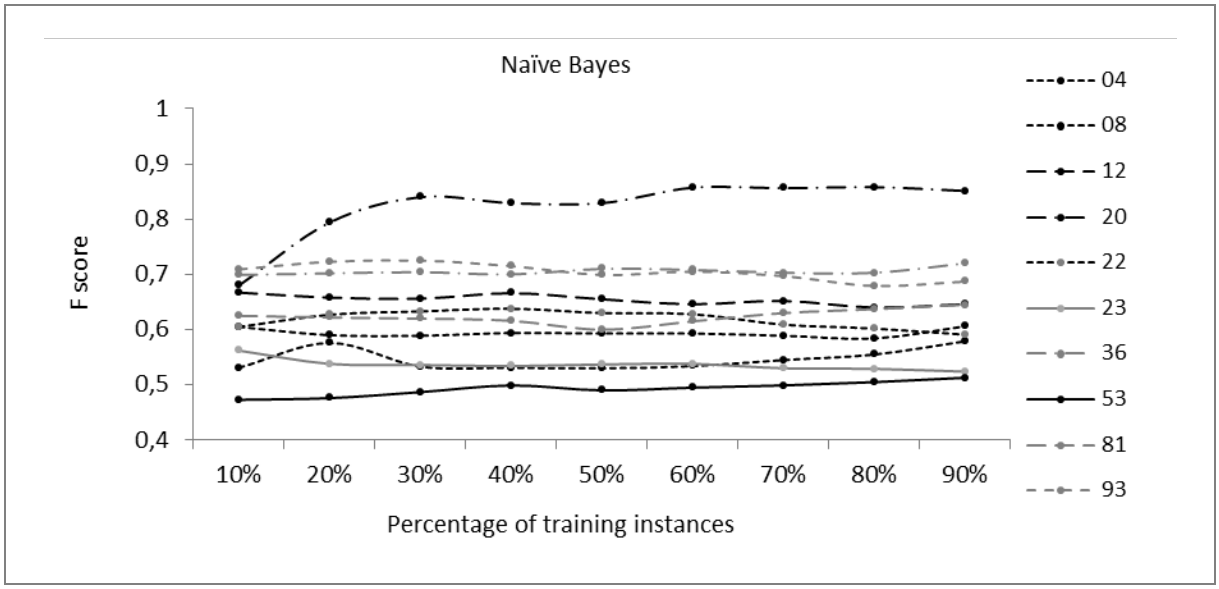

Figure 65 Naïve Bayes classifier F Score result with noise

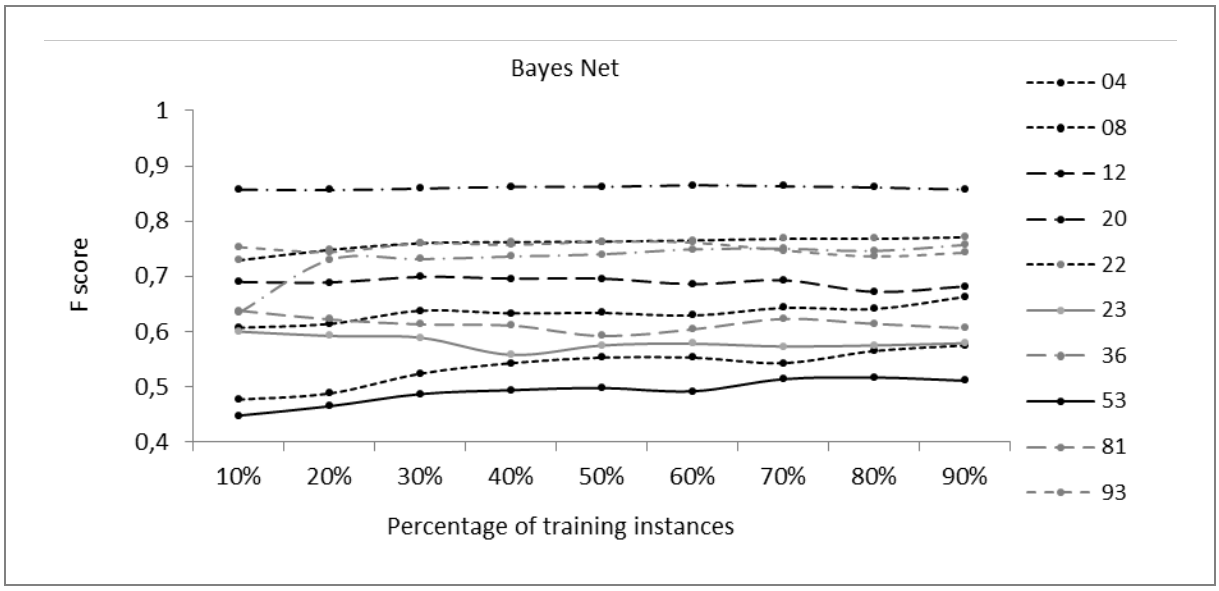

Figure 66 Bayes Net classifier F Score result with noise

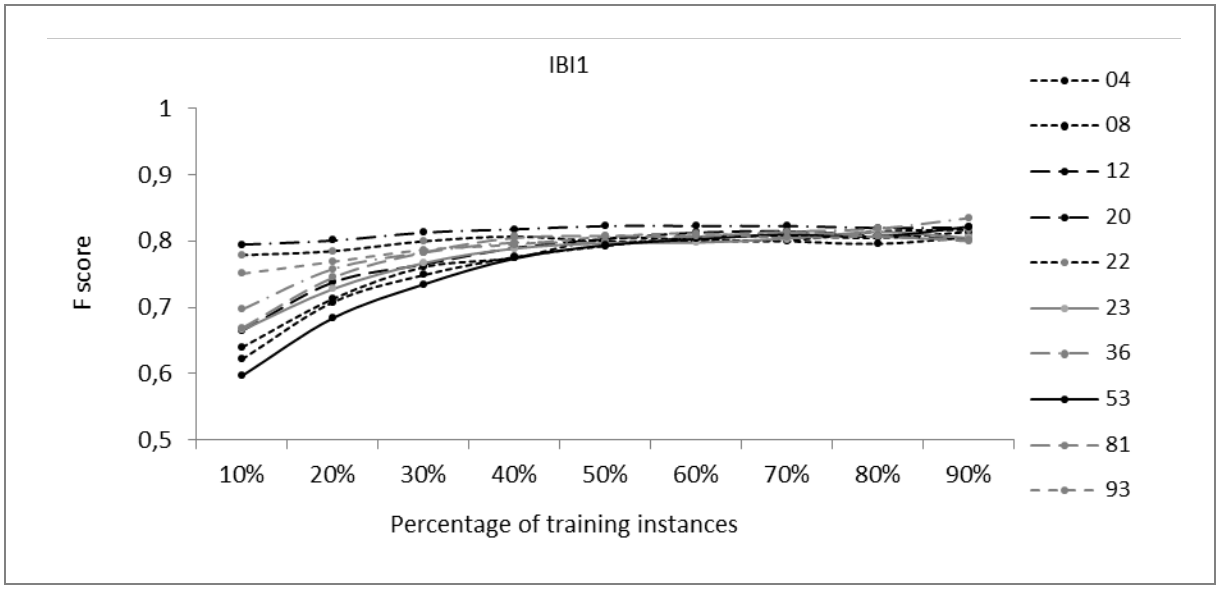

Figure 67 IB1 classifier F Score result with noise

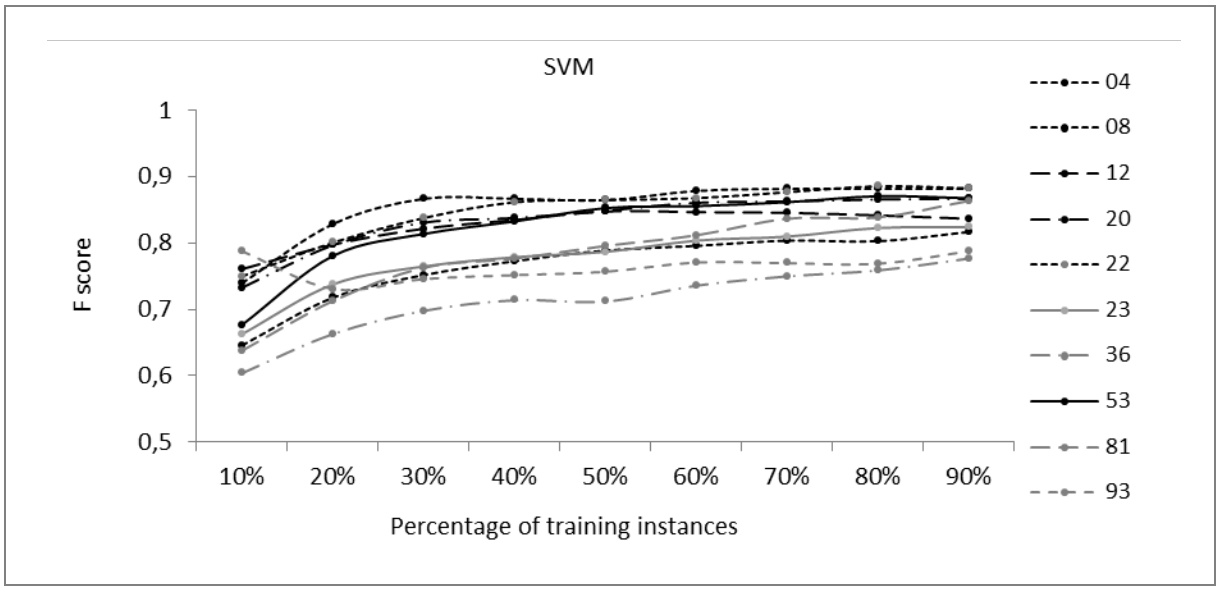

Figure 68 SVM classifier F Score result with noise

The results from the tests show that for all classifiers the **average accuracy rate** is 80% in all user datasets except for the Bayes family. The average variance and confidence interval are both below 2%. There is no over-fitting effect, and the maximum accuracy value is of 90%. The best performing classifiers are the Tree C4.5 and SVM with an average result of c. 83%, the Decision Table and IB1 with an average result of c.80% and last the Bayes family (Naïve Bayes and Bayes Net) with an average of 64% to 68% respectively. The classifiers that show higher resistance to incremental noise are the SVM and Decision Table that degrades 9.5% while the IB1 degrades 17% and the Bayes Family remain with similar values.

The case of the Bayesian family classifiers (Naïve Bayes and Bayes Net) is again different; their overall classification accuracy is in average 64% and 68% respectively, and it is not affected by the changing of the training size, they both have an average standard deviation of 0.1 in the accuracy results when increasing the training size and in average of 0.011 to 0.0054 respectively in the difference in the accuracy result when increasing the training size. Their accuracy result again fluctuates, now from 58% to 78% in average for both classifiers.

We obtain a consistence performance in average on the performance of the algorithms in the **users** with our noiseless tests analyzed in "Section 5.2.1 Classification Results". The user on which the classifiers perform the best in average is again user 20, with a result of c. 86%, and the user on which the classifiers perform the worst in average is again user 53 with a result of c.70%.

In all cases, the **F Score** overall results are similar with the accuracy rates of the classifiers, dropping 0.10 points in average (13%) to 0.75. consistent with the classification results (the same happens with the average variance and confidence interval, which are both below 2%.)

The Bayes family classifiers show the lowest result of c.0.6-0.65 in average, but their decrease related to noiseless datasets F Score values is lower than the average, only 9% in

average, which confirms their robustness to irrelevant features (as explained in “Section 3.2.1 Description of the algorithms”.) Decision Table, SVM, and IB1 classifiers have an F Score value in average of 0.8, and the Tree C.4.5 performs slighter better with a value of 0.84 in average. The ranking of the F Score values in average differ from that of the noiseless F Score results, when adding noise the classifier that has the biggest decrease is the IB1 (consistent with the accuracy results) that drops 18%, while the remaining decrease an average of 13%.

The results show that when including noise in the datasets, the classifiers in average misclassify 25% of the dataset’s instances as belonging to one class when in fact they belong to another. The `recall` values are lower than the `precision` values in the Bayes family algorithms, which indicate that the type of error these algorithms make is of leaving out instances that belong to a class.

5.3 Performance results

We go one step further and test how our system would work in a real case scenario. Our goal is to find an appropriate balance between the algorithm’s performance and its scalability by analyzing the time it takes to build the model given a certain amount of data used for training, how well it works if the patterns of the user change along the way, and the impact in context inference of all the variables we have defined in our system.

5.3.1 Changing the behavior patterns

We create a new dataset combining two users, with the goal to create a disruption in the user’s pattern by combining two datasets of different users which will allow us to test the impact in classification and the classifier’s robustness to changes in behaviors and patterns once it has been trained. We choose users 04 and 53 for our task since they both have very different usage of the phone: one uses the phone mainly for communication (voice and specially text) and the other for internet (email usage). This difference in uses translates into different patterns that will help to increase randomness in the new dataset.

This combination results in a new dataset in which we change the number of instances we select to create the combined dataset, we first choose all instances in both datasets, then we reduce the number of instances in the 53 dataset, selecting only 20%, maintaining all the instances in the 04 dataset, then we do the opposite, we decrease to 20% the amount of instances we choose from the 04 dataset and maintain all the instances in 53 dataset. In both cases we select the first instances in the dataset. With these different combinations of the datasets into a new one we are simulating the amount of time the user would be behaving in one particular usage pattern before it changes to another one. We train the classifiers using the first 20% instances of the resulting dataset, since we have previously

seen that in average, this is the amount needed to train the classifiers. Table 25 shows the details of these new datasets.

Combined datasets		Instances		
User	**Name**	**Total**	**Train**	**Test**
04 full & 53 full	0453full	11976	2395	9581
04 full & 53 20% instances	04full53partial	7215	1443	5772
04 20% instances & 53 full	04partial53total	7157	1431	5726

Table 25 Additional dataset used in the experiments

The graphs in Figure 69, Figure 70, and Figure 71 report the classifying accuracy of each algorithm over the datasets when changing the usage patterns.

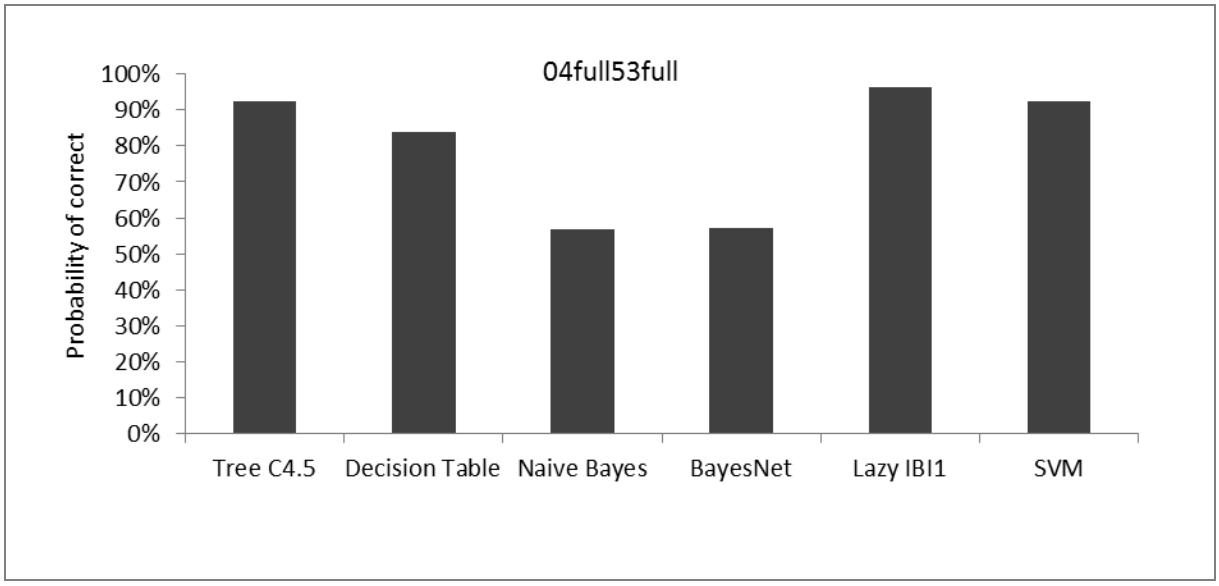

Figure 69 Accuracy results trained over 20% of the dataset of combined user 0453 full datasets

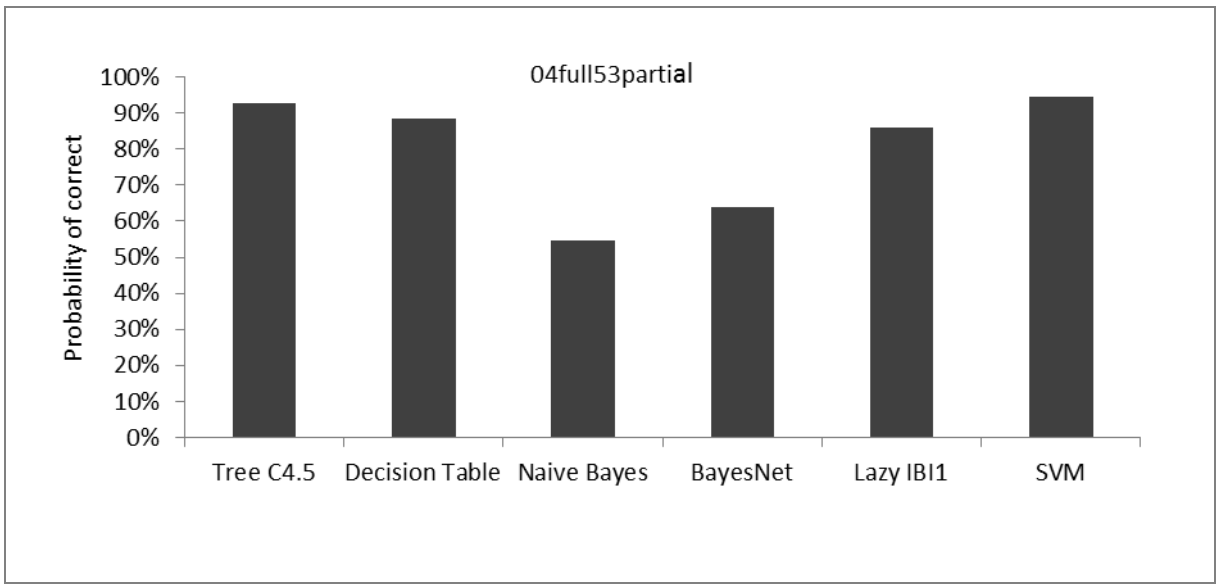

Figure 70 Accuracy results trained over 20% of the dataset of combined user 04full53partial

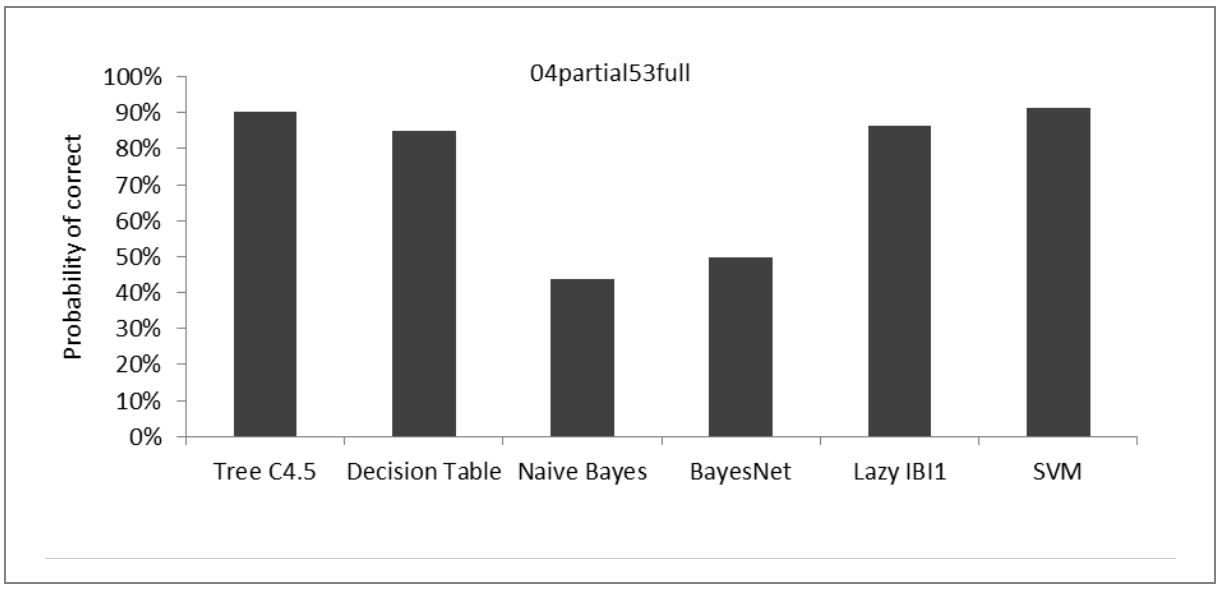

Figure 71 Accuracy results trained over 20% of the dataset of combined user 04partial53full

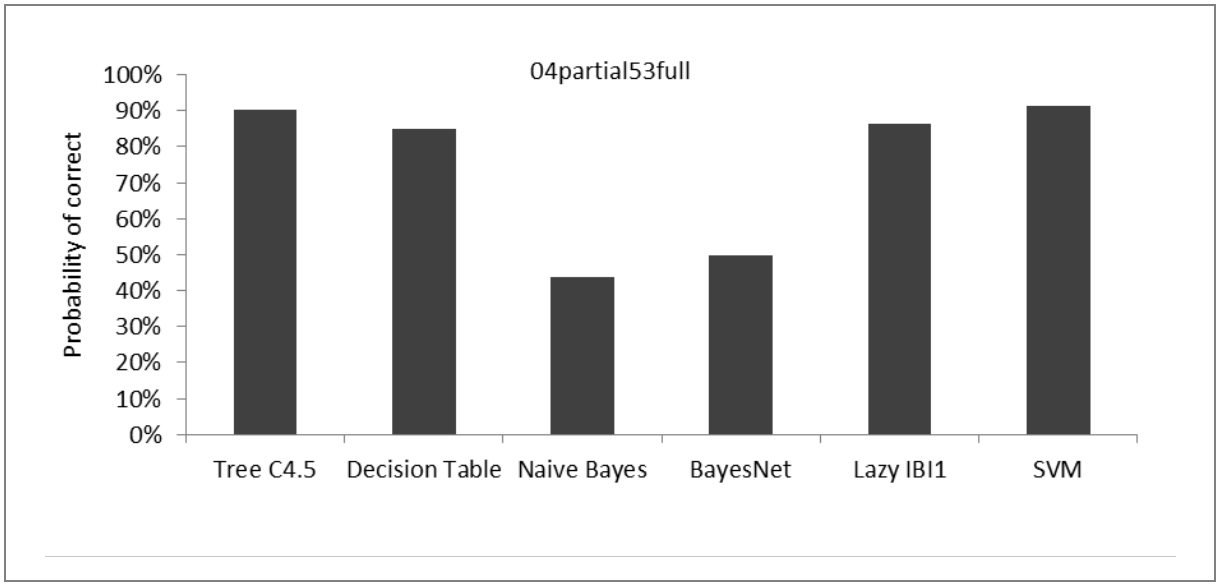

Figure 72 Accuracy results trained over 20% of the dataset of combined user 04partial53full

The graphs in Figure 69, Figure 70, and Figure 71 report the F Score of each algorithm over the datasets when changing the user's patterns.

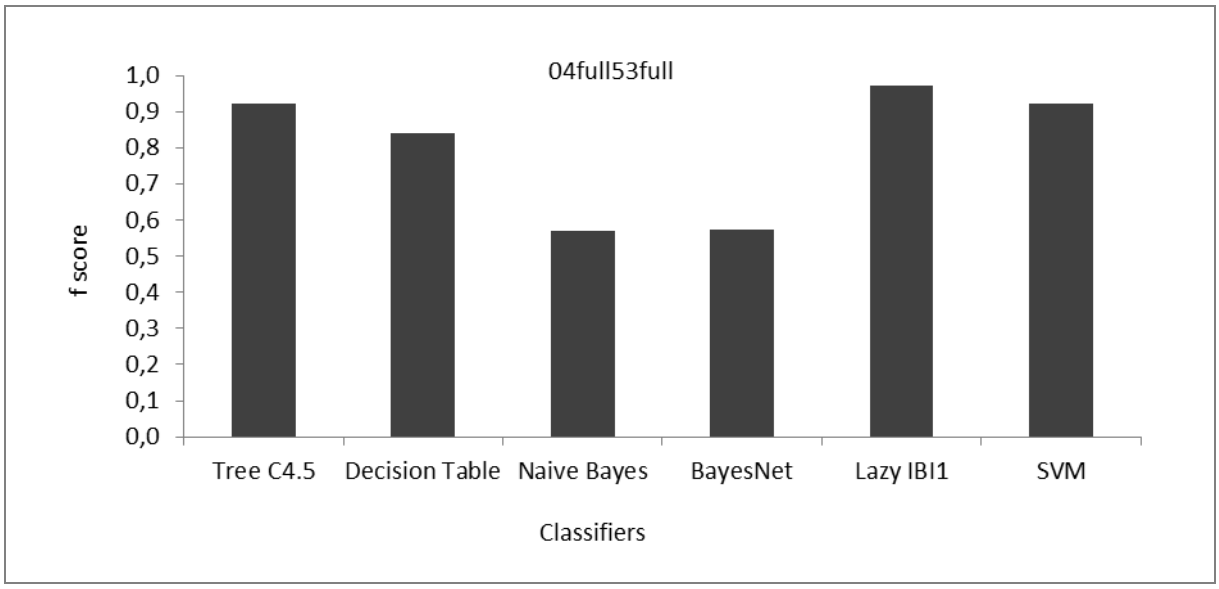

Figure 73 F Score results trained over 20% of the dataset of combined user 0453 full datasets

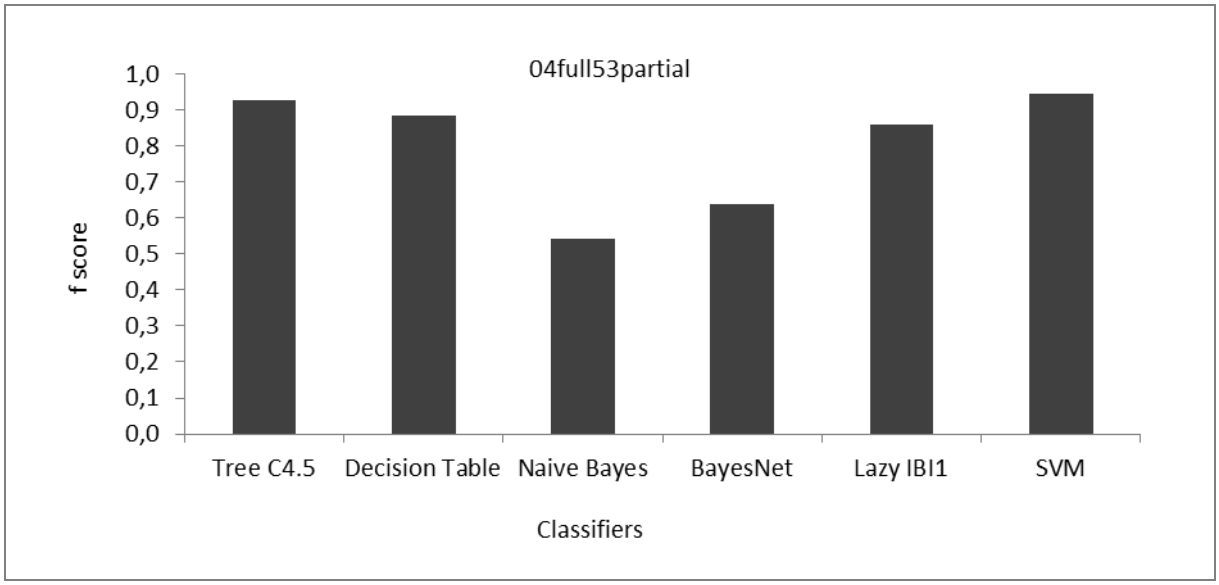

Figure 74 F Score results trained over 20% of the dataset of combined user 04full53partial

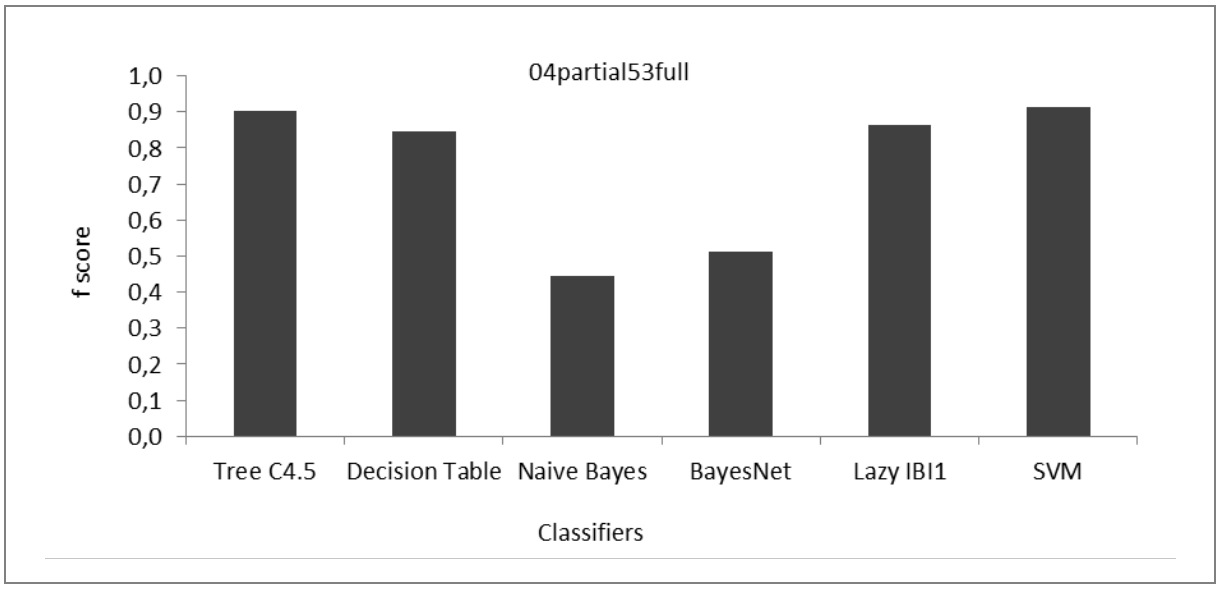

Figure 75 F Score results trained over 20% of the dataset of combined user 04partial53full

As expected, the **accuracy results** are worse than those obtained when testing the classifiers over user 04's dataset alone, but their overall performance despite the change in user patterns is still high, with an average result of almost 80%. The average variance is below 2% with an average confidence interval below 5%.

Each algorithm reacts differently to the changes in usage patterns, the Bayes-based classifiers are the ones most affected by it, with an average decrease in accuracy of c.17%, and dropping their accuracy rate below 60% in average.

On the other hand, the IB1 classifier is not affected negatively by the change in user's pattern, and it even improves when the dataset size increases to the combination of both full datasets 04 and 53.

The rest of the classifiers' **accuracy** decreases 4% in average, maintaining a good accuracy result of c.90% in average, and showing an improvement in the accuracy rate when the dataset size is reduced (i.e. we only take 20% of the instances of user 53's dataset).

The **F Score** results obtained are aligned with the accuracy ones, and again as expected, they are worse than those obtained when testing the classifiers over user 04's dataset alone, but better than the ones obtained for the user 04 when adding noise to the dataset. The overall F Score value is high despite the change in user patterns, and in average, c.20% of the instances in the dataset get misclassified. The variance and confidence values are aligned to the accuracy ones again, again with an average of 2% and 5% values respectively.

The difference in the F Score values when the user changes its pattern is aligned with the one in the classification accuracy. In general the values remain high and very similar to the noiseless results for user 04's datasets, except for the Bayes-based classifiers that again are the ones most affected by it.

The type of mistake these algorithms make is to misclassify instances as belonging to one class when in fact they belong to another, which is shown by their value of `Recall`, which is lower than the value of `Precision`.

Again the SVM classifier is the one less affected by the change in the user's pattern as it maintains a similar F Score value.

5.3.2 Runtime to build the model

We run our tests training the classifiers using the first 20% of the dataset again and present the results of the average runtime for each classifier in Figure 76, with the goal to help us determine out the right balance between the learning time the classifier needs to build a model versus the complexity and sophistication of a learner that has better results (Domingos 2012).

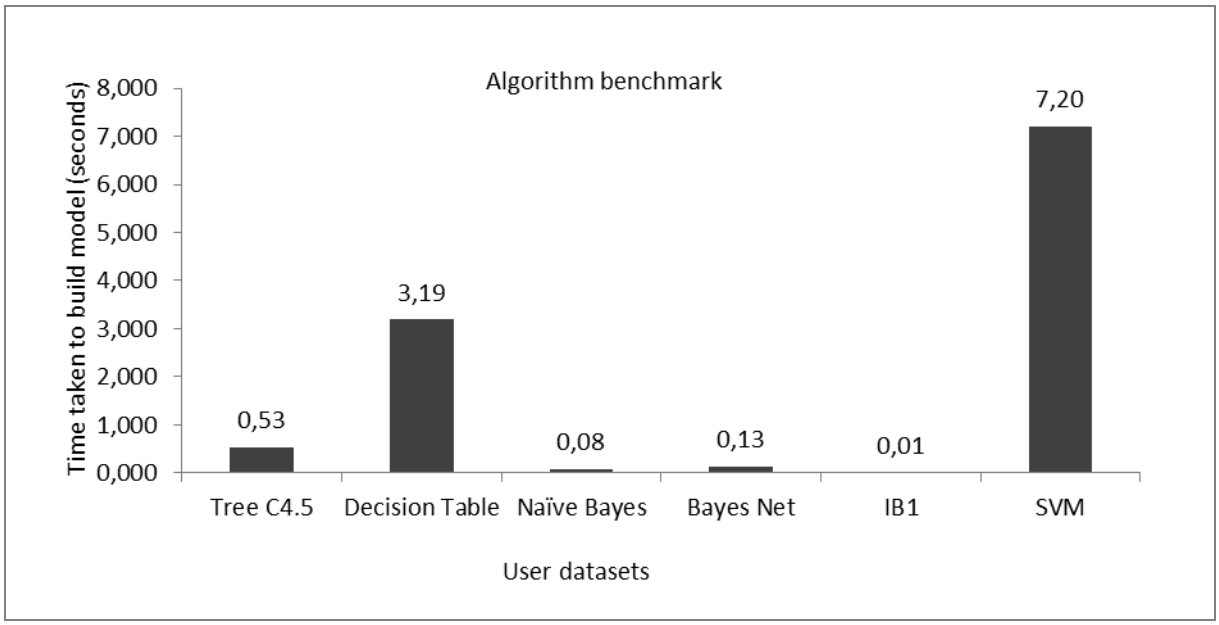

Figure 76 Classifiers' runtime benchmark to create the model

The results from the tests show that for all classifiers except the SVM and the Decision Table, the **average runtime** is 0.2 seconds, in all user datasets with an average variance below 5% and an average confidence interval below 10%.

We explain each average runtime,

- The time the Tree C4.5 algorithm needs to create the model is in average 0.5 seconds, ranging from 0.24 seconds in user 20's case to 0.71 seconds in user 12's case.
- The time the Decision Table algorithm needs to create the model is in average 3.17 seconds, ranging from 2.63 seconds in user 53's case to 3.69 seconds in user 23's.

- The time the Naïve Bayes algorithm needs to create the model is in average 0.08 seconds, ranging from 0.07 seconds in user 23's case 0.10 seconds in user 81's case.
- The time the Bayes Net algorithm needs to create the model is in average 0.13 seconds, ranging from 0.07 seconds in user 04's case to 0.16 seconds in user 23's case.
- The time the IB1 algorithm needs to create the model is in average 0.01 seconds, ranging from 0.010 seconds in user 08, 20, 53, and 93's case to 0.014 seconds in user 12's case.
- The time the SVM algorithm needs to create the model is in average 7.21 seconds, ranging from 5.25 seconds in user 22's case to 9.18 seconds in user 12's case.

Overall, the **fastest** classifier to build the model is the IB1,which is known to require almost zero training time because the training instance is simply stored, and the Bayes family classifiers, that are also known for a short computational time for training since they train very quickly requiring a single pass to count frequencies (Kotsiantis 2007). The **slowest** one is the SVM algorithm that takes more than double to build the model than the rest of the classifiers. This difference in time frame is due to the fact that the latter has to calculate all the separating hyper planes and compare them recursively until it finds the appropriate one while the IB1 classifiers stores the whole dataset at once to apply the formulas to calculate if the instances belong or not to the output class.

5.3.3 Attribute impact on prediction

In addition to the prediction accuracy, it is necessary to optimize the classifiers during testing by ensuring that the datasets used contain the appropriate features. We analyzed the impact various attributes have on the accuracy results when predicting context classes to determine the advantages in using them to train the algorithm (Kohavi 1995).

Choosing the right features helps us maximize the classification accuracy for an unseen test set; we define the optimal feature subset with respect to a particular induction algorithm by accounting for its nature (heuristics, biases, and tradeoffs). An optimal feature set can be defined as the following: given a classifier function, $f(x)$, and a dataset, D, with features $X_1, X_2, \ldots X_n$ and a certain distribution over the label instance space, the optimal feature subset $X(opt)$ is a subset of the features that maximizes the results of the classifier over the dataset. The primary issue we face is that we usually do not have access to the underlying label space distribution and, hence, must estimate the classifier's accuracy from the data (Kohavi 1995); we propose a methodology for this in Figure 77.

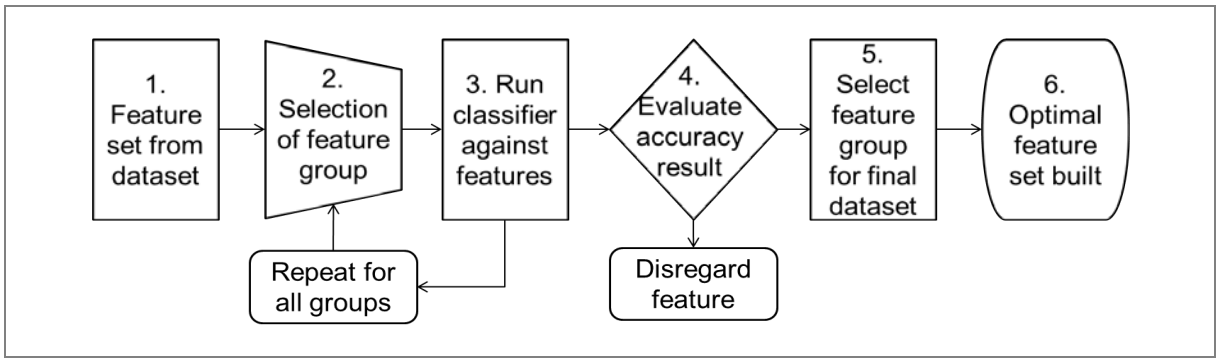

Figure 77 Methodology to select the optimal feature set

It is important to consider that there is no single unique optimal subset because it might be possible to achieve the same accuracy using different combinations of set features, which usually happens when two features are perfectly correlated and can therefore replace each other (Kohavi 1995).

We now focus on each attribute group and test the accuracy of the algorithms when using the variables grouped and individually. We note the classification accuracy results to understand if there is any specific attribute or sub-attributes that have more relevance in the class. We test the attributes described in Table 18, where we described each context variable and the values it can take, which was the basis to define the context matrix (Figure 39) that we are using as input for our classifiers. For example, attribute group `Communication` is made of all the communication related sub-attributes, i.e. voice call, text message, etc. shown in Figure 36.

We use 20% of the dataset to train the algorithm again. We test all of the sub-attributes in the attribute groups and present the average results in Figure 78, Figure 79, Figure 80, and Figure 81.

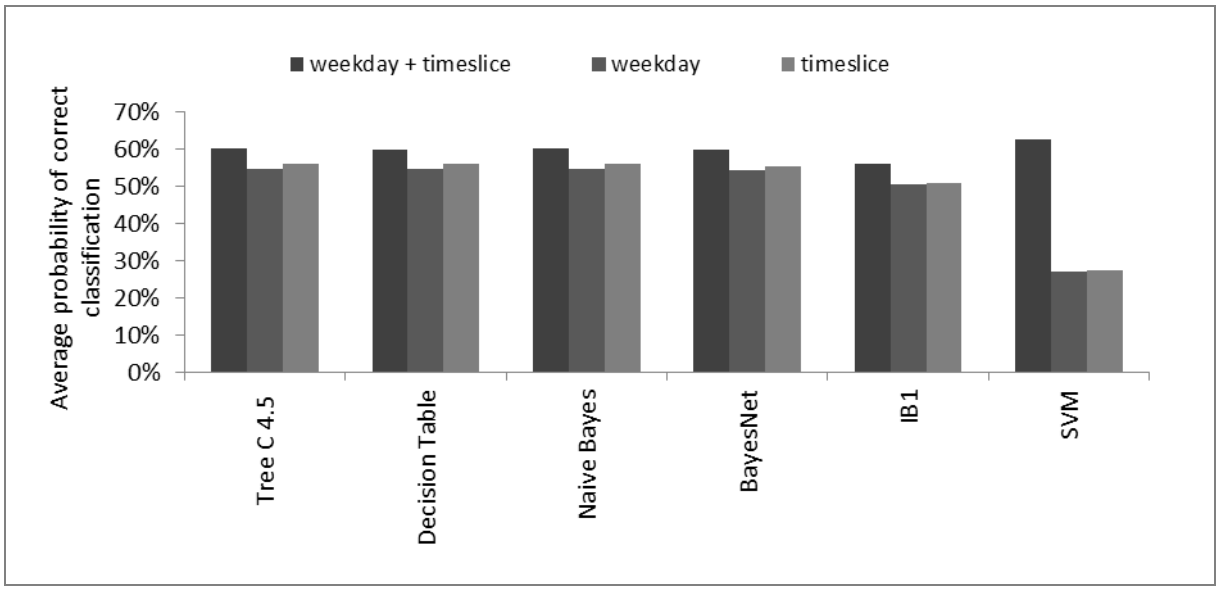

Figure 78 Relevance of weekday + time-slice sub-attributes

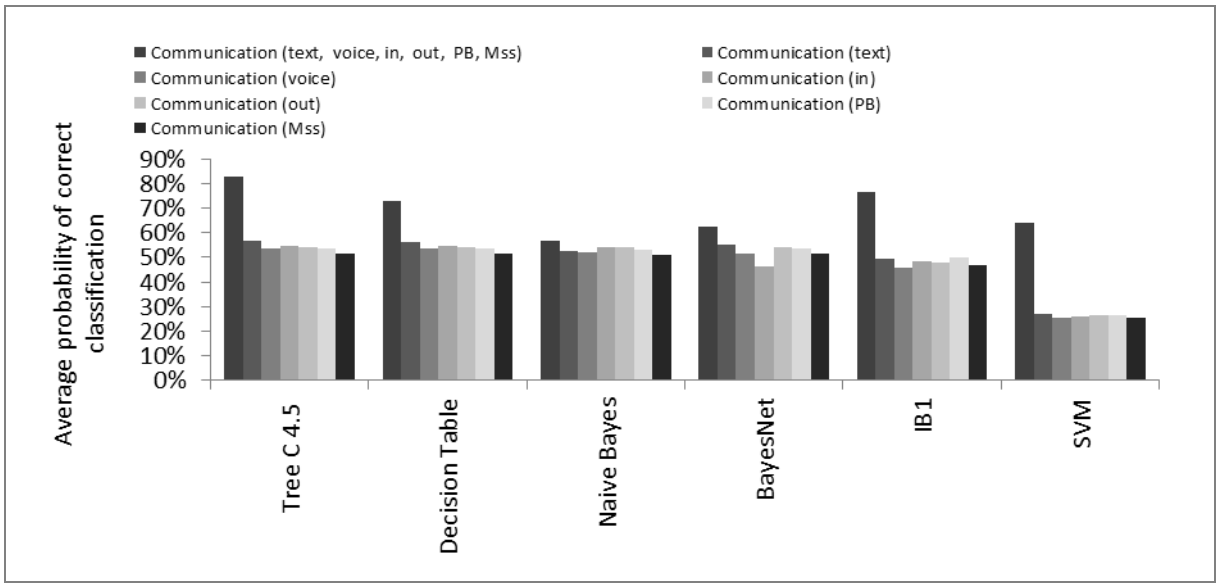

Figure 79 Relevance of communication sub-attributes

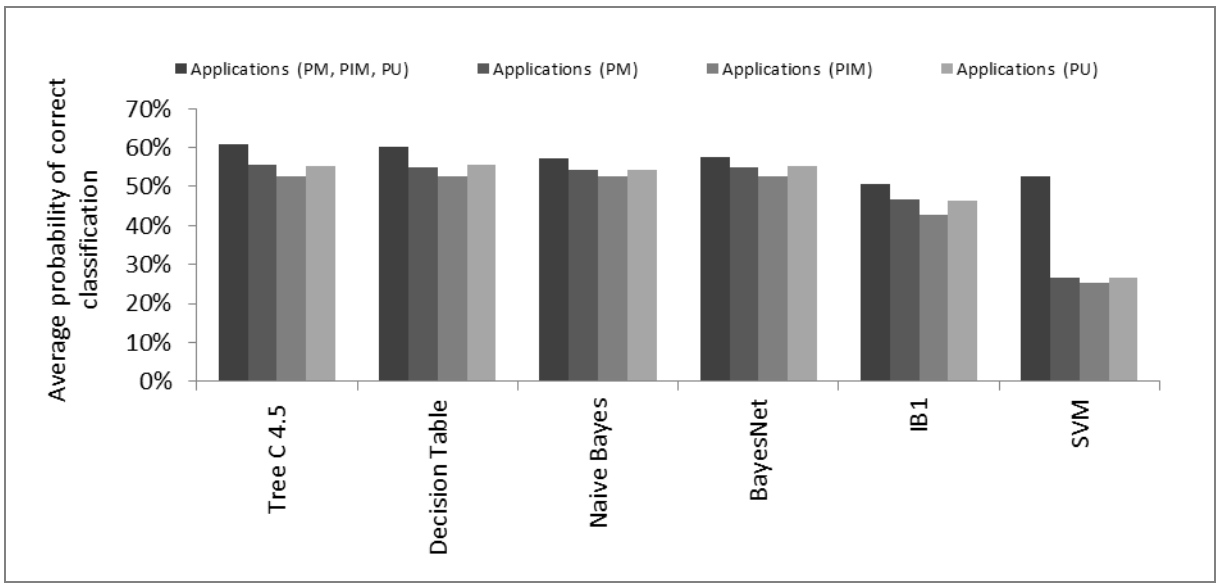

Figure 80 Relevance of application sub-attributes

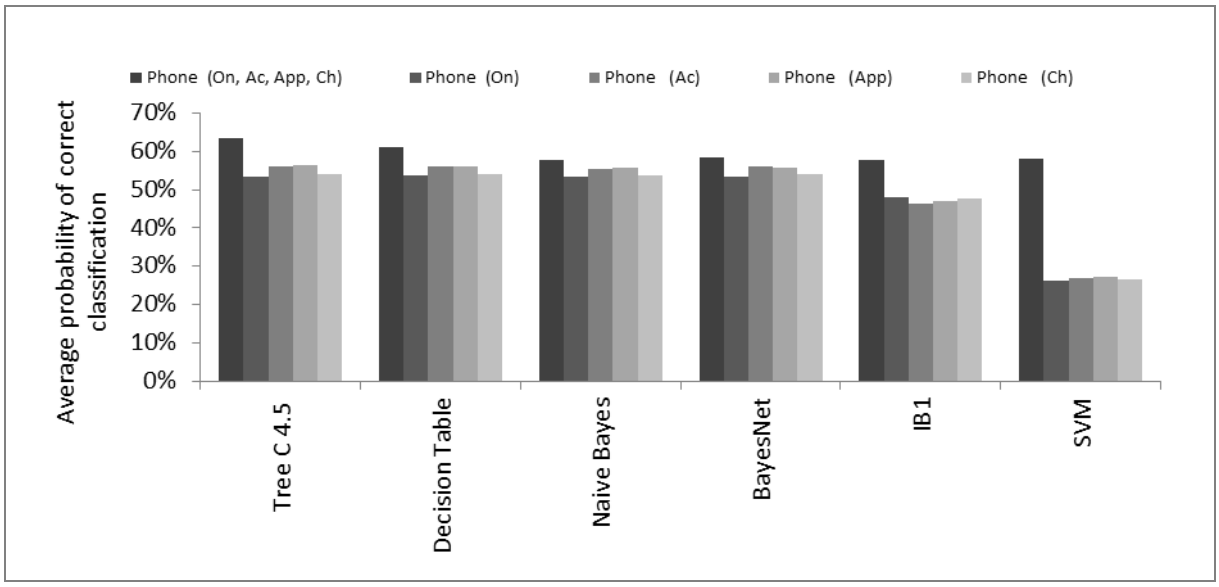

Figure 81 Relevance of phone sub-attributes

In all cases, when the variables from one group are considered together, rather than individually, it has a higher impact in the accuracy level of the classifiers. This situation is especially noticeable in the SVM classifier, on which the average accuracy with grouped variables doubles the average accuracy with individual variables. This is due to how this algorithm works to derive context paring variables together to classify them.

The same phenomena occurs for the communication context variables on all classifiers, which shows that communication variables are interrelated and on their own they do not have a meaning in our context scenarios on which a context is derived by an action from the user, and communication is a combination of incoming and outgoing calls and messages.

The communication context group is the group that has a higher impact in general in context classification, which is due to the type of usage that mobile devices had on the period when the data from the Reality Mining project was captured, 2004/2005, in that period phones and smartphones were voice centric and therefore their main usage was for communication. Moreover, at that time instant messaging multiplatform applications such as “WhatsApp” had not yet been introduced in the market and text messaging was the main alternative to voice.

5.4 How to use classifiers in our model

The experimental results we have performed provide a checkpoint for our mobile context model and inference proposal and allow us to evaluate the best performing classifier. When recommending a classifier to use in our mode, we need to balance between the classifier’s accuracy and F Score without overfitting, its resistance to noise and the amount of instances and time it takes it to build a model. We will prioritize resistance to noise and lower

overfitting when ranking the classifiers to select the one we will recommend to use in our model, as more sensors are included in smartphones, the amount of noise in the context variables captured will increase.

Context classification

The context states we have proposed as our classes, "Working", "Relaxing", and "Moving" have shown to be flexible enough to work in environments with noise and with behavior pattern changes enabling the algorithms to output good classification results above 80%. Inferring these context state with this accuracy result is very positive because it means that by adding the contextualization model proposed, applications can make use of the proposed context states which have many uses such as improving user interfaces or adapting productivity applications in such a way that if the context is guessed wrong, it does not impact negatively the usability experience. A simple example would be if the context derived is "Working", then the phone ringer can be turned to silent so that it does not disturb the user, and in the contrary, if the context derived is "Moving", the phone ringer can be turned to very loud to allow the user to hear it while on the move. Another example can be if the context derived is "Relaxing", the list of frequently used applications that the device displays can be ordered by leisure applications first. In all these examples, if the context is guessed wrong, the action that would follow by mistake does not have excessive importance in the user experience even if it is not the appropriate one for the context the user is in, moreover, the context's action rules could even be overridden manually by the user to create personalized exceptions.

In Chapter 7 "Mobile search: a real life example" we explain in detail how our context can be used in an application such as a search engine.

Best performing classifiers

We present a benchmark of the overall results for the classifiers when testing with different users in Table 26.

Accuracy with and without noise		**Runtime seconds to build model**		**Noise impact in performance decrease**	
High [100% – 80%]	Tree C4.5 SVM	Fast [0 – 1[	Naïve Bayes Bayes Net Lazy IB1	Small [0-10%]	SVM Naïve Bayes Bayes Net
Medium]80% –70%]	IB1 Decision Table	Medium [1-2.5]	Tree C4.5	Medium]10-20%]	Tree C4.5 Decision Table
Low]70% –50%]	BayesNet Naïve Bayes	Slow [2.5+']	Decision Table SVM	Large [20%+]	IB1

Table 26 Algorithm results overview

The most accurate classifiers in general are the Tree C4.5 followed by the SVM, with the later showing to be less impacted by noise than the rest of the best performing classifiers.

The least accurate classifiers were the Bayes family, Naïve Bayes and Bayes Net, which is due to their basic assumption that all features are independent of each other (Cheng and Greiner 1999) which is not the case in a mobile phone usage environment. However, they have shown to have a high tolerance to noise, decreasing in average c.6.5% in their overall performance.

In all cases, we have seen that the datasets have an average of nine months of data compiled, which is the timeframe that the users stay in the project and coincides with the period of time on which their phone is active as we confirmed when analyzing the datasets. The classifier uses 20% random instances of the dataset to train itself and learn the classification model.

Optimal attribute subset

Overall, our tests show the context variable groups with their sub-attributes: Weekday + Timeslice, Location (H, W, X), Communication (text, voice, in, out, PB, Mss), application usage (PM, PIM, PU), and Phone (On, Ac, App, Ch), have a significant impact in context prediction (in average c.60%) and have shown to have a higher impact when combined into groups rather than when used individually. This is more evident in the case of the communication context variables, in which the accuracy results are doubled when the variables are combined, reaching an average value of 80% which is the highest impact in overall context classification as well.

We therefore recommend including all the sub-attributes when building the context vectors and to prioritize communication variables over the others, i.e. ensure that all (or most of) the instances used to train the classifiers have communication values in them.

Classifier recommendation

Our simulation test results indicate that Tree C4.5 and SVM classifiers have the best performance combinations. We recommend the SVM classifier as the best option for our model because it combines good accuracy despite the noise level, which is a very important factor in mobile devices, since the data captured by sensors usually have a lot of noise. Due to its slowest time to build the model, we suggest to place the algorithm's logic in our architecture outside the mobile device, in the cloud, for those applications that are internet based and that require an execution time lower than 1 second.

In our architecture, we recommend to use the model we have created in our tests while training the classifier, and then to compile six to nine months of mobile usage, and retrain the classifier using the same methodology (train it over 20% random instances from the dataset and test it over the remaining 80%). Since it is a large amount of data, we suggest performing the training task outside the mobile device, i.e.in an external server or in the cloud, and once the contextualization model is created, send it back to the contextualization application that is running on the mobile device so that it can be updated to classify the new

phone usage instances that are read into context states that will be used to contextualize applications.

5.5 Conclusions and discussion

In this chapter, we have illustrated how to apply the mobile context-awareness model and architecture phase 2 to context inference. We first defined a framework for the context data model to infer the context states in such a way that they can be learned by machine learning algorithms such as classifiers. We empirically evaluated and compared six classifiers, Tree C4.5, Decision Tables, NaiveBayes, BayesNet, Lazy IB1, and SVM, using ten real-life datasets chosen from the Reality Mining project ("Section 3.3.2 Data description"). We evaluated both their performance rate and runtime after training them using over 10%, 20%, 30%, 40%, 50%, 60%, 70% 80%, and 90% of the dataset. We test as well the impact of noise in the performance of the classifiers, adding incremental and radical noise to them. We also individually analyze the attribute groups to understand their impact on class prediction.

Our experiments show that the best performing algorithms without noise are the Tree C 4.5, IB1 and SVM classifiers. When adding noise to the datasets we obtained consistent performances in average with the noiseless results, except for the SVM classifier that is less affected by noise and IB1 that is more affected by noise. The algorithms that have performed the worst in their accuracy when classifying the datasets have been the Bayes family classifiers, with the Naïve Bayes performing worse than the Bayes Net classifier. Their accuracy in average does not improve when the training size increases, although they do not show a large decrease when adding noise to the datasets.

Some classifiers have shown to overfit the model when they are given a large amount of instances to train, i.e. above 60% of the dataset. We therefore recommend training the classifiers with the minimum necessary amount of instances in the dataset in order to achieve a good accuracy level and avoid possible over-fitting situations, which we have seen is c.20% in average.

In all cases, the F Score results are aligned with the accuracy rates of the classifiers, and the precision/recall ratio is almost one in the noiseless data sets, while on the data sets that have noise the precision value is higher than the recall for the Bayes family classifiers, showing that they miss to classify instances in the overall class groups.

The nature of each user, the characteristics of their datasets and their phone usage patterns have an impact on the results of the classifiers. So for example, user 20 reports to use the phone at work and for personal uses, which is reflected in the amount of instances belonging to the working or relaxing classes, which are the majority of the dataset, this results in a consistent higher classification outcome than for the rest of users in all classifiers, with and without noise.

When changing the usage patterns, the accuracy results are still good, an average of 78%, which shows that our classifiers with the context states that we have defined are fit to be used over a mobile device on which usage patterns have seasonality changes inherent in them (i.e. uses of mobile devices in summer vacation are different than those in school or work) times, and different than those of season vacation such as Christmas for example.

The average runtime of the classifiers to create the model ranges from 0.01 to 7.21 seconds, in all cases, the fastest classifier to build the model is the IB1 and the slowest one is the SVM taking an average of 8 seconds overall. The difference in time frame is due to the fact that the latter has to calculate all the separating hyper planes and compare them recursively until it finds the appropriate one while the IB1 classifiers stores the whole dataset at once to apply the formulas to calculate if the instances belong or not to the output class.

We propose using the SVM as the primary algorithm for our model because it showed to be more robust and maintained the performance results even when adding noise to the datasets, and it has a fast performance (an average time of 6 seconds) and needs a small percentage of instances of 20% to train to achieve a good accuracy above 90% and 80% in average with and without noise.

When analyzing the attributes and their impact in the classification results accuracy, we found that in general it is better to use the variables combined into their groups, rather than individually, to increase the probability of correct classification. We have also found that the communication group variables have a higher impact in the classification result than the rest of the other variables, which coincides with the fact that all users report to use the phone mainly to call and text.

In the next chapter, we test our proposed method to enable the classifiers to infer future context states by including a step to predict the upcoming context signals that were previously used to build the context vector that is the classifier input as described in the architecture chapter. We test the prediction guess rate of the Markov model algorithms and explain how to use them to adapt to each user's specific patterns.

6 Predicting future context states

In this chapter, we explain how to predict context signals to create a context vector for classifying context states to predict future context states. This capability will allow us to test the proposed predictive feature in our context-aware architecture that allows the classifiers to infer future context states.

We analyze how to predict context signals in a mobile environment using the location and applications as examples based on a history of previous events with some type of inherent pattern. We perform an in-depth analysis of the results of this algorithm for the ten user traces from the Reality Mining project that we defined as the dataset traces in the tools and background chapter. We use the Markov Model on-line machine learning algorithm described in the tools and background chapter to predict the future position and usage of a mobile device in terms of its location and application use based on the previous location and applications used. This predictor is based on the assumption that the next event depends on a number of previously observed events. The size of the events seen in the past corresponds to the order of the algorithm. We present the prediction tests results for Markov Model algorithm using real data context variables available from the Reality Mining project.

Our goal is to determine how well suited Markov Models are for forecasting context states in a pervasive computing environment requiring an on-line prediction method where the learner receives one example at a time and must estimate the output before receiving the correct value (Cristianini and Shawe-Taylor 2000), as well as to propose some options to further improve its accuracy.

6.1 Test framework

We build a prediction algorithm based on learning techniques to infer the next context signal as part of a context vector that the classifiers will use to predict context states as part of phase 2 of the context-aware architecture we defined earlier. We use an on-line type of learning where all of the data are given to the learner on the go. We use the context variables defined earlier and manipulate the file formats to convert them into the sequential symbols expected by the Markov model algorithm for use in the prediction.

We analyze the accuracy of the Markov Model learning algorithm and how well it performs under the memory constraints of mobile devices. In our analysis, we account for the number of symbols needed for predictions, which is the algorithm order, and the amount of history required for the algorithm to have enough information about the user's patterns to achieve a successful prediction rate.

Datasets

We again repeat our tests using the 10 datasets we selected from the Reality Mining project. We conducted our experiments using two context variables, application usage and location. Table 27 shows a summary of reach user's dataset's information available.

User dataset	Days in project	Location		Application	
		instances	Different symbols	instances	Different symbols
93	277	80971	2691	12344	20
81	224	45094	1751	8606	19
53	296	34614	1744	12249	18
36	272	56174	2206	12869	20
23	277	72403	2840	12491	19
22	240	45546	2430	10508	18
20	293	72362	3137	20284	18
12	277	56458	3137	9838	20
08	256	45775	2794	10508	16
04	277	59132	1744	10728	19

Table 27 context signals data traces overview

We can see that there is not necessarily a relation between the number of days the user stays in the project, the number of instances the datasets has and the amount of different symbols. So for example, user 53 is the one who stays the longest in the Reality Mining project but user 93 has the longest dataset and user 23 has the largest amount of different symbols in the dataset. This is more evident in the application context signal case since there were a fewer number of applications available in the mobile phone than location cells, at the time when the experiment was done.

Test description

We ran a large batch of on-line prediction tests to predict both the location and application variables and report their results:

- Symbol window size: We test the algorithm's prediction results changing its order from 1 to 2 and report the prediction results.
- Result benchmark: We compare the results from the algorithm on the different user datasets and report its performance.
- Adding time: We adapt the algorithm filtering datasets by different time frames and analyzing the impact in performance.

Performance metrics

We determine the algorithm **accuracy** (Song, Kotz et al. 2006) by evaluating the guess rate as a percentage of the symbols recognized in the sequence of events, which is the fraction of events where the predictor correctly identified the next event after the trace. These data help us understand how well the algorithm works.

We analyze the **order** of the algorithm, which represents the size of the window of previous symbols. We change the order from 1 to 2 and evaluate the effects on the algorithm performance.

Finally, we also note the **entropy** value of the dataset, as it can be a good indicator of the amount of pattern there is on each user's data trace since it is a descriptor of randomness, therefore it may be a good indicator of predictability and performance of the predictor algorithm (Song, Kotz et al. 2006).

We use the standard definition of entropy, where the entropy, $H(X)$, of a discrete random variable, X, in an alphabet X, of symbols, x, with the probability of occurrence of a symbol x is $P(x)$, is shown in Equation 7 (Shannon 1948):

$$H(x) = \sum_{x \epsilon X} P(x) log_2 P(x)$$

Equation 7 Definition of entropy

We also relate the prediction results with the dataset's **instances/symbol ratio**, which relates the number of instances and the amount of symbols following Equation 8 for this.

$$dataset\ size\ (number\ of\ instances) : dataset\ symbols\ (number\ of\ different\ symbols)$$

Equation 8 Definition of instance to symbol ratio

These values will help us understand the impact of the dataset's nature on the prediction results.

Implementation constraints

When implementing and running the on-line algorithm for the context signal prediction we find several restrictions related to the amount of instances required to obtain accurate predictions, the size of the tree the algorithm builds and the impact of the noise level in the results. We measure the **number of nodes** required to build the algorithm and the **amount of instances** needed to start predicting.

Programming tools

We ran our programs on a Dell Latitude 4200 laptop with Windows 7 OS. We programed our context model module and learning algorithms using the Python open source software

issued under the GNU General Public License because it is a good, simple programming language that can be used for a wide variety of tasks with low maintenance costs that runs on several platforms, including mobile phones, which will simplify the migration of our model to a real environment. Python is a high-level general-purpose programming language that can be applied to many different types of problems (Python.org). Python has a variety of functions embedded in its library that allows us to program learning algorithms that take historical data of the simple context state and predict the next state. We use Python version 2.7.2, released on June 11th, 2011, which was the latest version available at the time we programmed our learning algorithms. We can see a full description of our implementation of the Markov Model in Appendix B: Program description.

6.2 Context signals prediction results

We analyze and report the results from an empirical evaluation of a basic, unsupervised on-line algorithm, the Markov Model that we implement. Our goal is to understand how well it performs in a real environment using the Reality Mining project by running it through the chosen data traces. First we report the cumulative prediction accuracy and then zoom into each user's dataset performance results.

6.2.1 Overall prediction results

The graphs in Figure 82 and Figure 83 report the distribution of the percentage of instances with a specific correct prediction for Markov Model algorithms. We calculate the accumulated average accuracy over all the instances in our ten datasets, as we increase the instance size by intervals of 100 up to 34600 in the location signal case and 8600 in the application signal case. We choose 34600 and 8600 as end values since those are the common number of instances that all datasets have. At each interval, we analyze the accuracy level the algorithm produces for each user. So for example, when the instance size is 2000, the algorithm outputs a prediction rate of: c.30% for user 93 and c.80% for user 53. We repeat this process for the whole amount of instances. We then calculate the accumulated average prediction rate of the classifier for each user, which we then add up from an instance size point of view and relate it to each percentage guess rate.

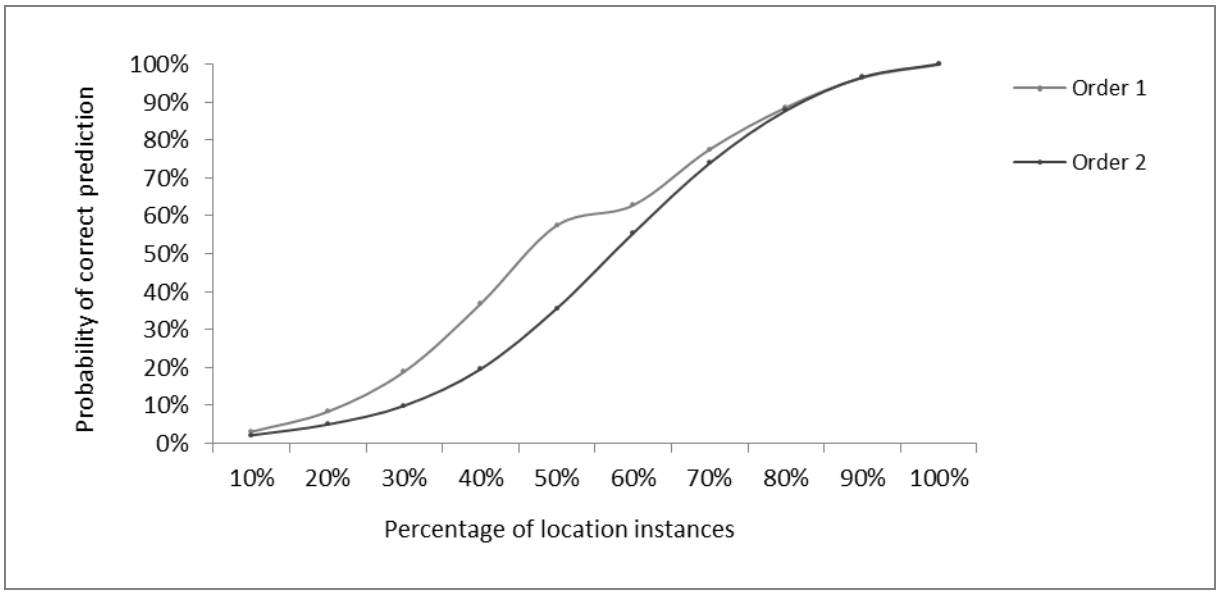

Figure 82 Accumulated accuracy of Markov predictor for location context signals

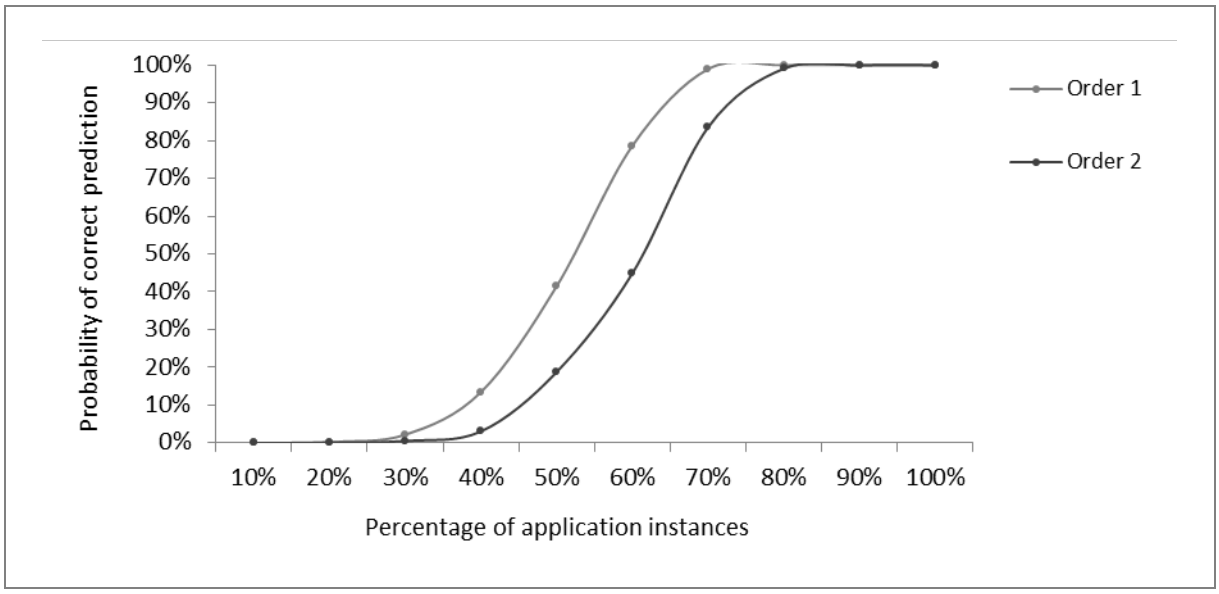

Figure 83 Accumulated accuracy of Markov predictor for application context signals

The distribution of processed instances of the total of context signals with the corresponding probability of correct prediction for Orders 1 and 2 drawn in these figures show that the probability of correct prediction is not fully linked to size of the instances processed and that for different processed signals size we get the same values. So for example, for 24220 location instances and for 6020 application instances or less (70% of their respective dataset), the probability of correct prediction of the next location symbol the algorithm achieves in average is of c.75% for Order 2 and c.80% for Order 1, while the probability of correct prediction of the next application symbol the algorithm achieves in average is of c.85% for Order 2 and of c.95% for Order 1. There is a difference in the performance when choosing Order 1 versus Order2 that is more accentuated for applications context signals. In

the next section we explore the prediction results for Order 1 and Order 2 on both context signals' datasets, analyzing the impact of the order in the results.

6.2.2 Performance results per user

We run each user's datasets independently through our implementation of the Markov Model predictor and report the results for every 100 events (instances) processed, benchmarking the prediction results, the number of nodes of the algorithm and the entropy value both for the location and application contexts. The context size (order of the algorithms) is set to both 1 and 2.

Algorithm results

Figure 84 and Figure 85 show the average probability of correct prediction for each user datasets for Order 1 and Order 2 for location and application context signals, while Figure 86 and Figure 87 show the average number of nodes for each user datasets for Order 1 and Order 2 for location and application context signals.

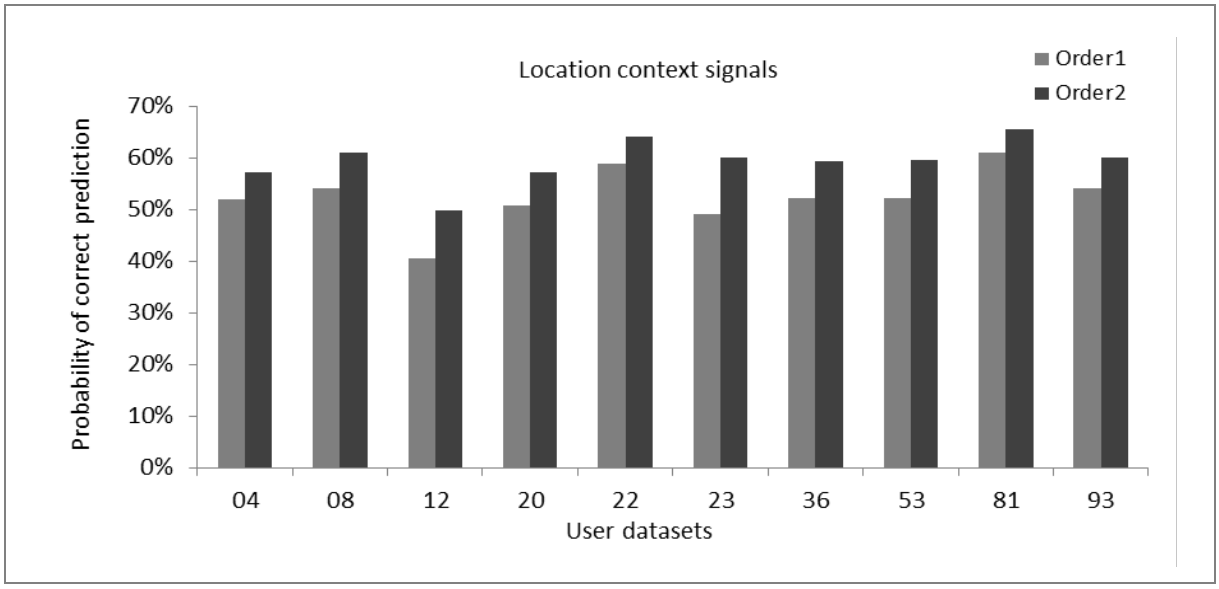

Figure 84 Markov Model prediction rate for location context variables

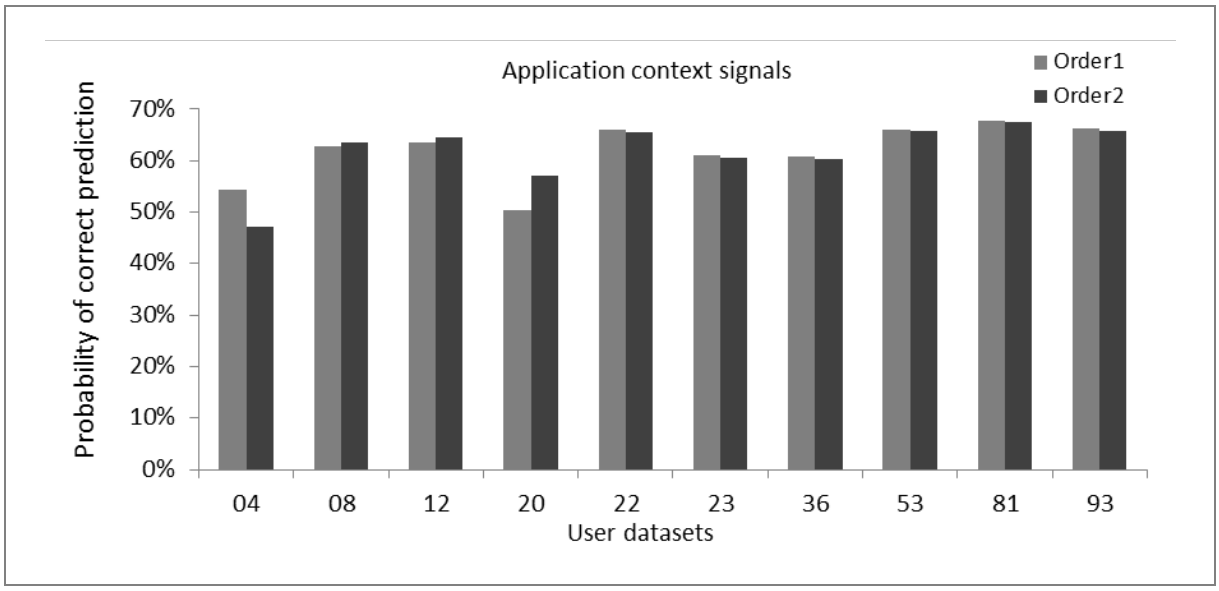

Figure 85 Markov Model prediction rate for application context variables

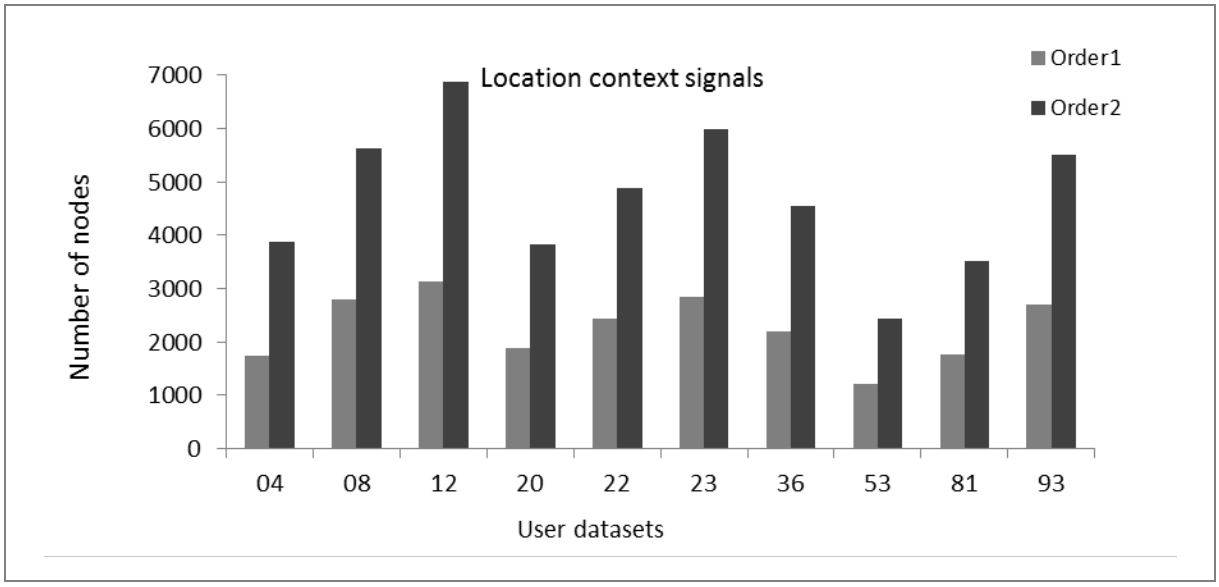

Figure 86 Number of nodes the Markov Model uses when predicting location context variables

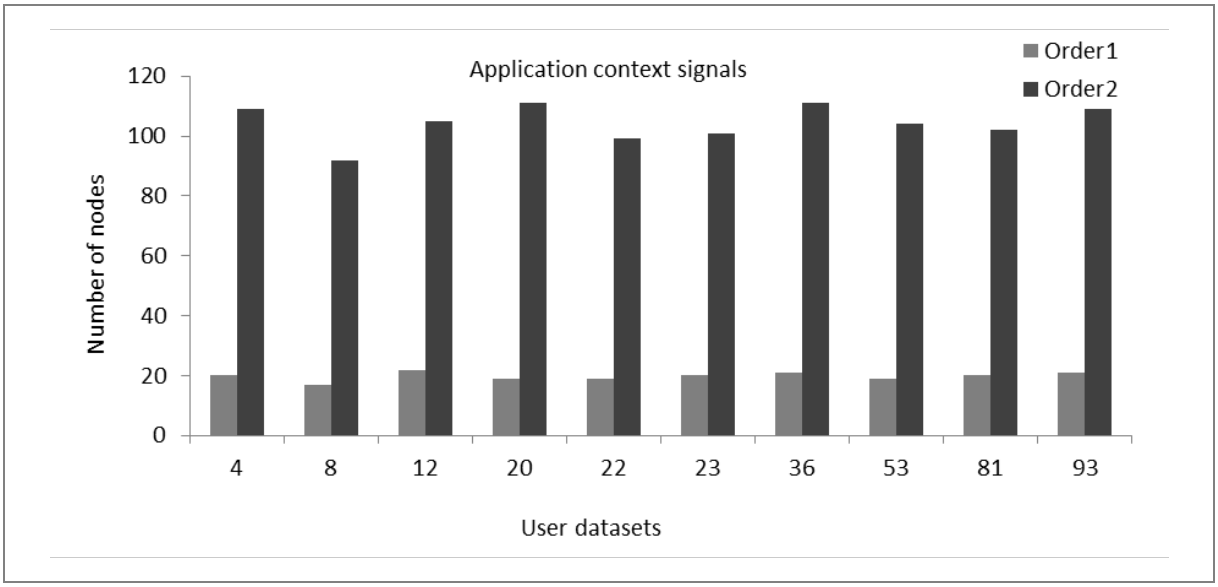

Figure 87 Number of nodes the Markov Model uses when predicting application context variables

The overall individual probability of correct prediction show that Order 2 out predicts Order 1 results in all cases for the location user datasets and on most cases for the application user datasets. Choosing a different order when running the algorithms will impact the prediction results depending on the nature of the patterns of the context signals analyzed rather than their size. For example, if we use Order 2, the guess rate will be based on the combination of three symbols that happen together, i.e. the amount of times that the symbol to be predicted follows the two symbols previously seen together (see the description and example in "Section 3.2. Machine learning algorithms" for further detail.) In the case of applications context signals, we would have an a priori set of twenty elements, shown in Figure 88, that can be combined into sequences of three, but nevertheless, there will be some combinations that will happen more often and others that may never happen, and this also changes from user to user, depending on the pattern usage that each person has.

Menu	Camera	PIM	Phone
Browser	Calendar	File explorer	Contacts
Messages	Configuration	Clock	Calculator
Notepad	Multimedia	Converter	Bluetooth
WiFi	App Manager	To Dos	Profile

Figure 88 Application menu from the mobile devices used in the Reality Mining project

In the location context signal case, the a priori set of elements is almost impossible to estimate, since it is based on the network topology defined by the mobile carrier which is

unknown to us: the placement of the phone towers and their antennas signal strength that will make mobile devices to attach from one to another as people move.

The order of the algorithm conditions the number of nodes the algorithm will need to build the tree. It significantly increases in Order 2, on the location user datasets case by an average of 52% and on the application user datasets by 81%. This increase will impact the memory consumption needs, which is limited in a mobile phone and it is not what we are looking for in our architecture design, as we explained in "Section 4.5. What our architecture proposes".

Users nature analysis

On the Reality Mining project, the users report information about the regularity and predictability of their schedule as we saw in ("Section 3.3.2 Data description"). On this section, we calculate the ratio between the amount of instances of the datasets and the different number of symbols, the entropy each user has and the existing correlation between those values.

Figure 89 and Figure 90 show the instances/symbols ratio per user dataset calculated using Equation 8.

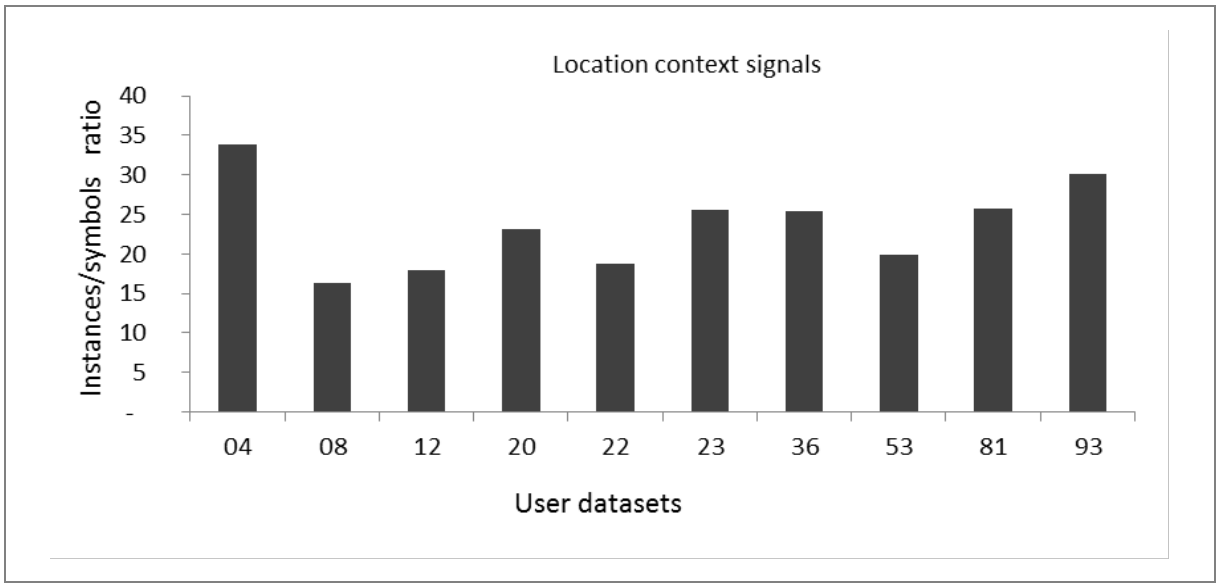

Figure 89 Instances/symbols ratio per user for location context signals

All location context signals datasets have a similar ratio from 15 to 25 with two exceptions; one is user 04 with a ratio of 34 and user 93 with a ratio of 30. This is due to the fact that user 04 has the largest amount of instances in its dataset (80971) and user 93 has the lowest amount of different symbols in its dataset (1744). Both users claim to use the mobile device mainly for personal communication, which implies that the locations from which they use their phone are limited.

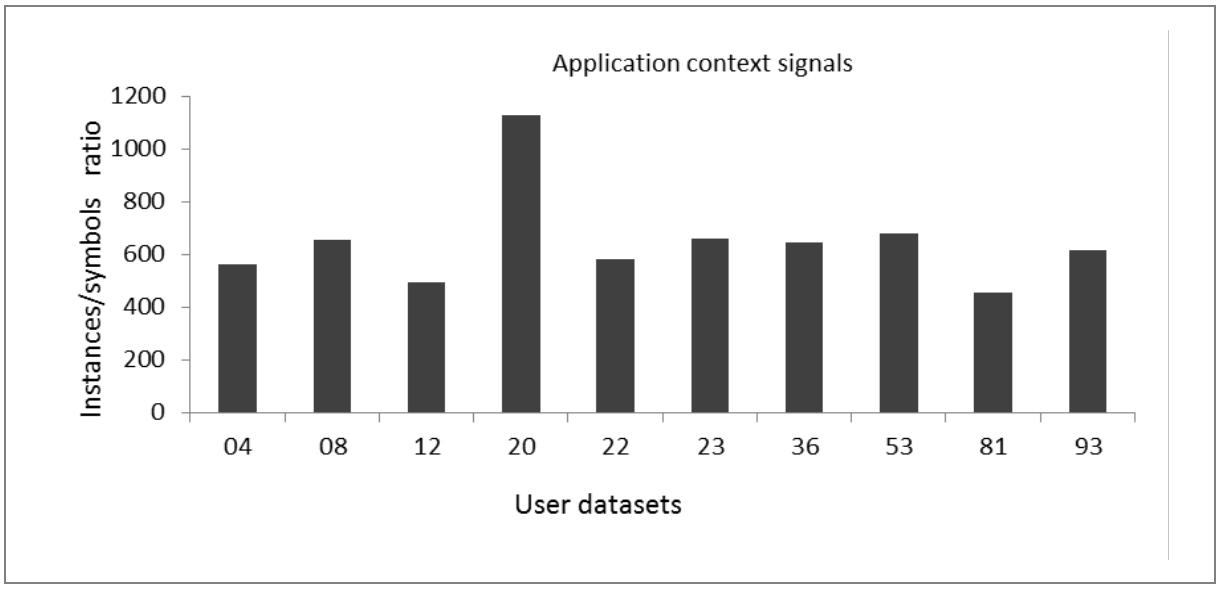

Figure 90 Instances/symbols ratio per user for application context signals

Since the amount of different symbols is limited in the application context signals case, the instance/symbol ratio values are all in the range of 550 to 700 except for user 20, which has an instance/symbols ratio of almost 1200. This is due to the fact that this user reports to being very active using the phone both for work and personal uses sending text and email often, in fact, this user has the largest amount of instances in its dataset (more than 20000) and stays the longest time in the project (293 days) out of the ten chosen users for the tests, as we see in Table 27.

Figure 91 and Figure 92 show the amount of entropy each user datasets have which we calculated using Equation 7.

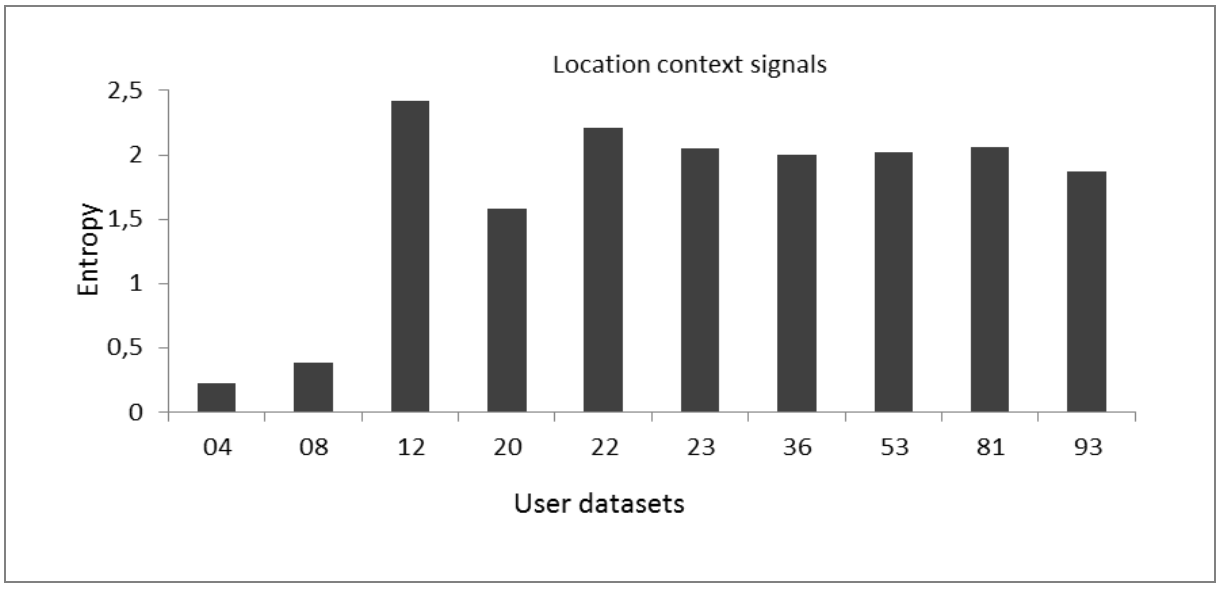

Figure 91 Entropy per user on location context signals

The entropy level of each user is different, being the lowest level that of user's 4 and 8, we know that these two users have in common that they both use their phone mainly for personal communication, which might influence the location from which they make this calls. The lowest entropy level corresponds to user 04, which is also the dataset with the fewest amounts of different symbols for the location context signals, as we see in Table 27.

The highest entropy corresponds to user 12, who in the Reality Mining survey reported to have a somewhat regular schedule but to be very predictable and to actively use the phone to email, text and call both for personal and professional purposes, which means that the phone was connected from many different locations. This user also has one of the lowest ratios of number of instances versus different amount of location symbols in the dataset (18) despite staying a long time in the project (277 days).

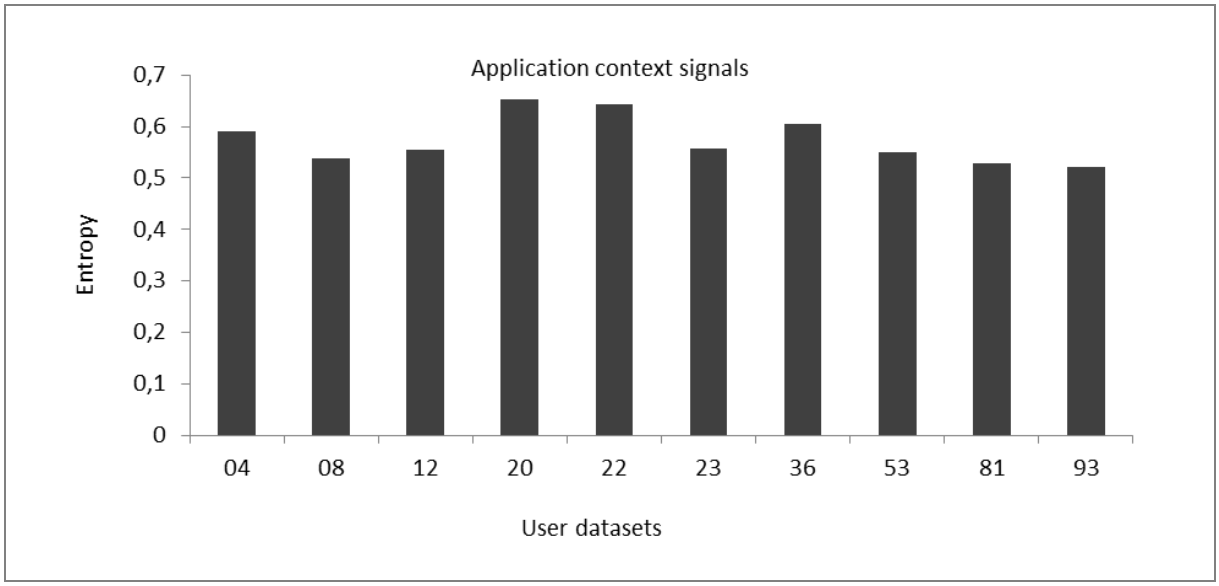

Figure 92 Entropy per user on application context signals

Again the entropy level of each user is different, being the lowest level on this case user 08, which also had low entropy for location context signals as we saw in Figure 91. This entropy level is aligned with the user's description of having very regular schedule and being very predictable, as well as with the fact that the user report to use the phone mainly for personal communication purposes and that there are only 16 different application symbols on the dataset, which is the lowest value in the project, as we can see in Table 27.

The highest entropy corresponds to user 20, who in the Reality Mining survey reported to have a somewhat regular schedule but to be very predictable as well as to actively use the phone both for work and personal uses sending text and email often.

Correlations analysis

Figure 93, Figure 94, Figure 95, and Figure 98, show the relation between the instance/symbols ratio and the average probability of correct prediction shown earlier in Figure 84 and Figure 85.

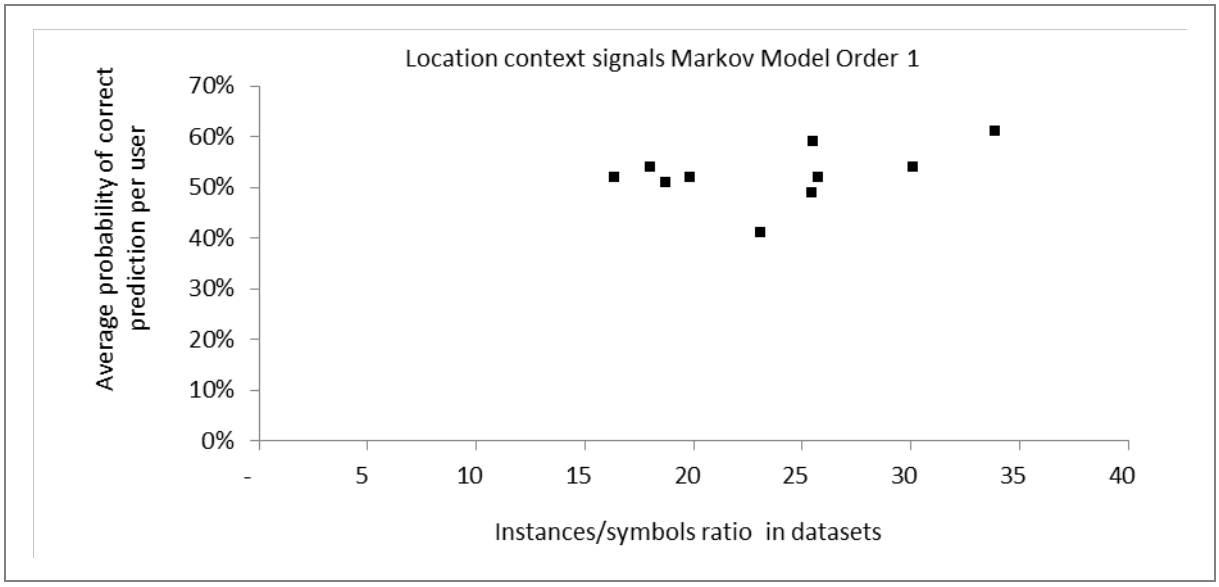

Figure 93 Correlating Markov Model Order 1 prediction for location datasets with instance/symbols ratio

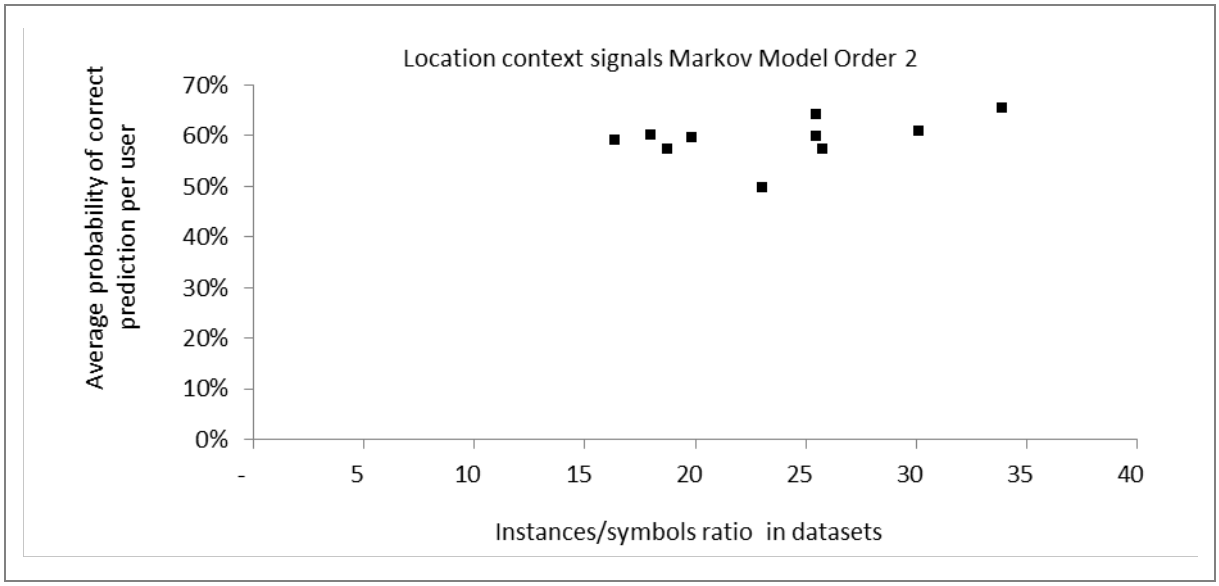

Figure 94 Correlating Markov Model Order 2 prediction for location datasets with instance/symbols ratio

When correlating Markov Model Order 1 and Order 2 prediction's on the location datasets with the instance/symbols ratio we see that although the best prediction rate is achieved over the highest ratio of instance/symbols, there is not always a direct relation between the instance/symbols ratio and the prediction achieved. For example, a ratio of c.25 instance/symbols, the average probability of correct prediction is c.45% for Order 1 and

c.50% for Order 2, while for an instance/symbol ratio lower than 20 the average probability of correct prediction is c.10% higher (c.55% for Order 1 and c.60% for Order2).

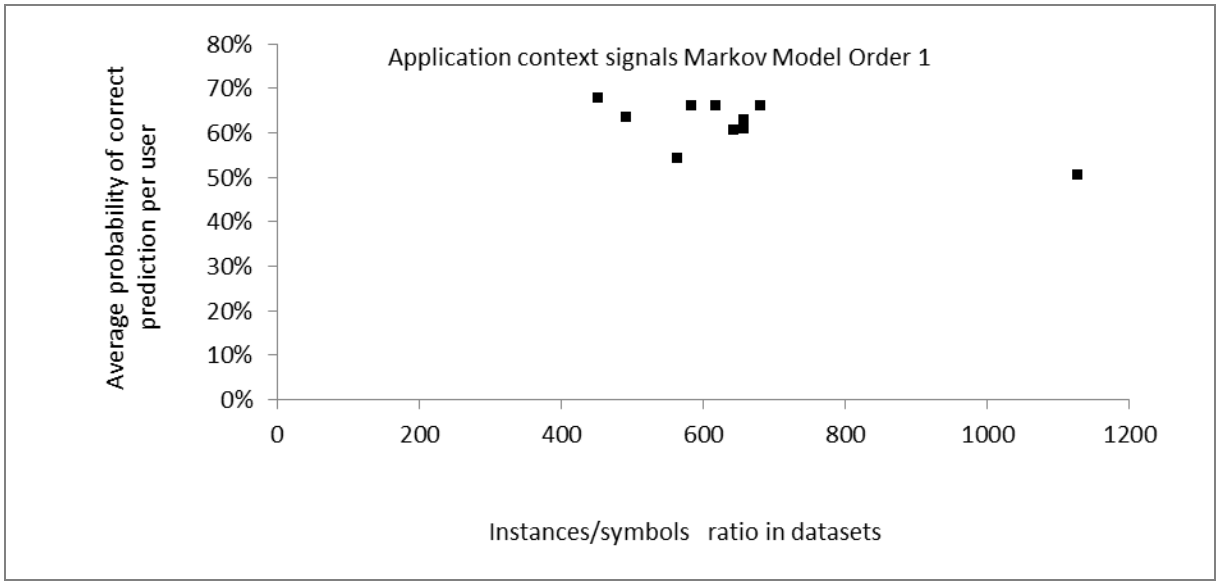

Figure 95 Correlating Markov Model Order 1 prediction for application datasets with instance/symbols ratio

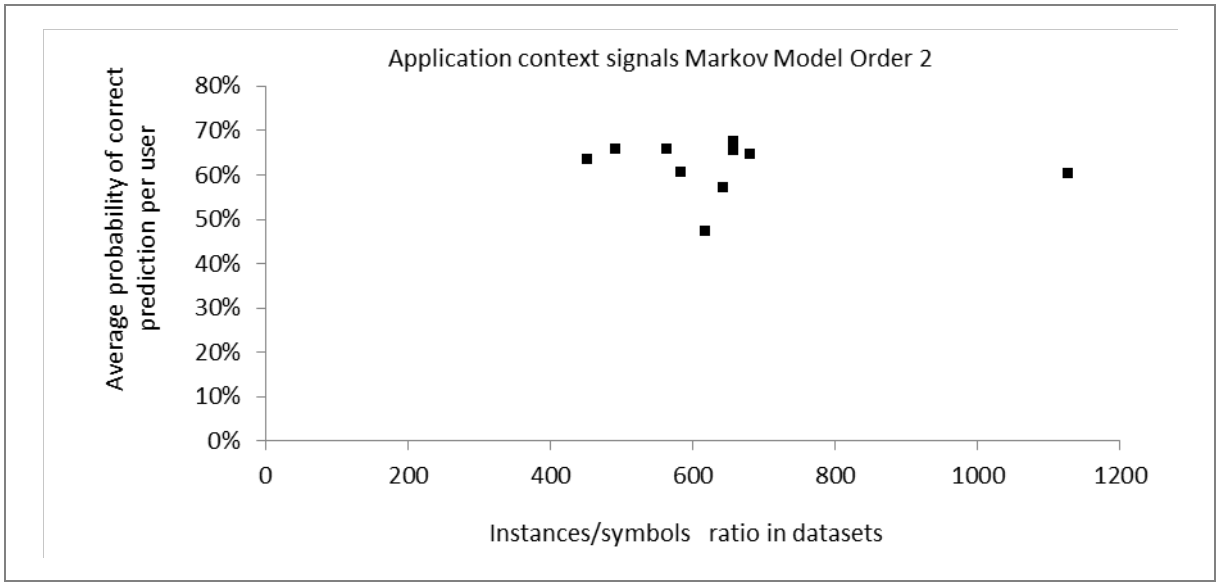

Figure 96 Correlating Markov Model Order 2 prediction for application datasets with instance/symbols ratio

When correlating Markov Model Order 1 and Order 2 prediction's on the application datasets with the instance/symbols ratio we observe a similar result that the location datasets. Again there is not a direct link between a higher instance/symbol ratio and the average probability of correct prediction both for Oder 1 and Order 2.

Figure 97, Figure 98, Figure 99, and Figure 100 show the relation between the entropy value and the average probability of correct prediction shown earlier in Figure 84 and Figure 85.

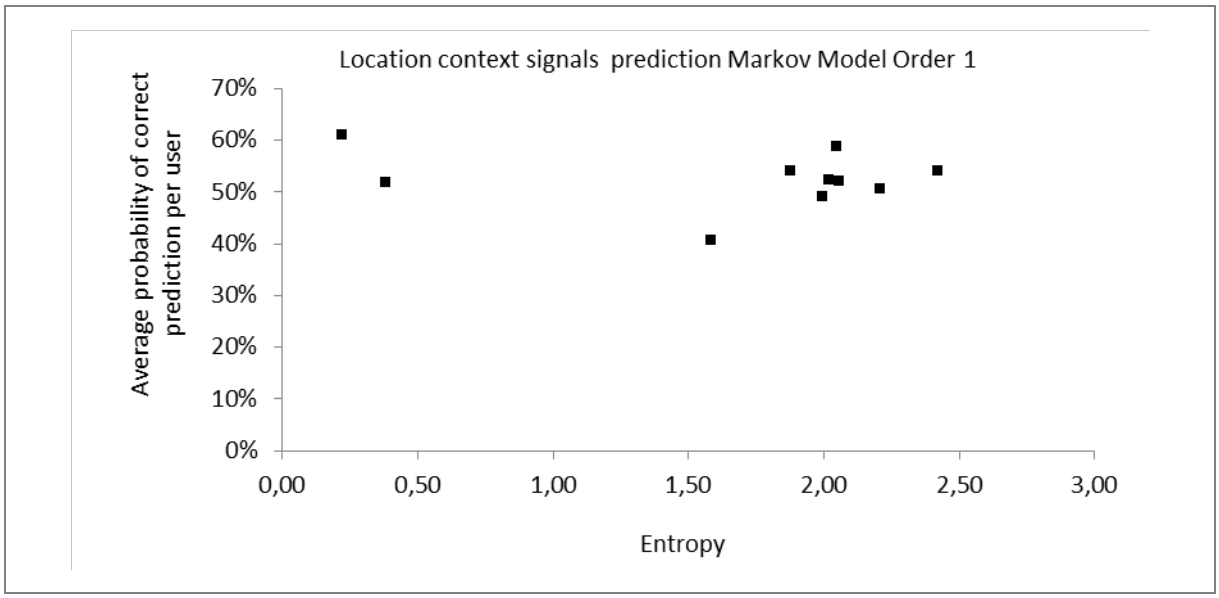

Figure 97 Correlating Markov Model Order 1 prediction for location datasets with entropy

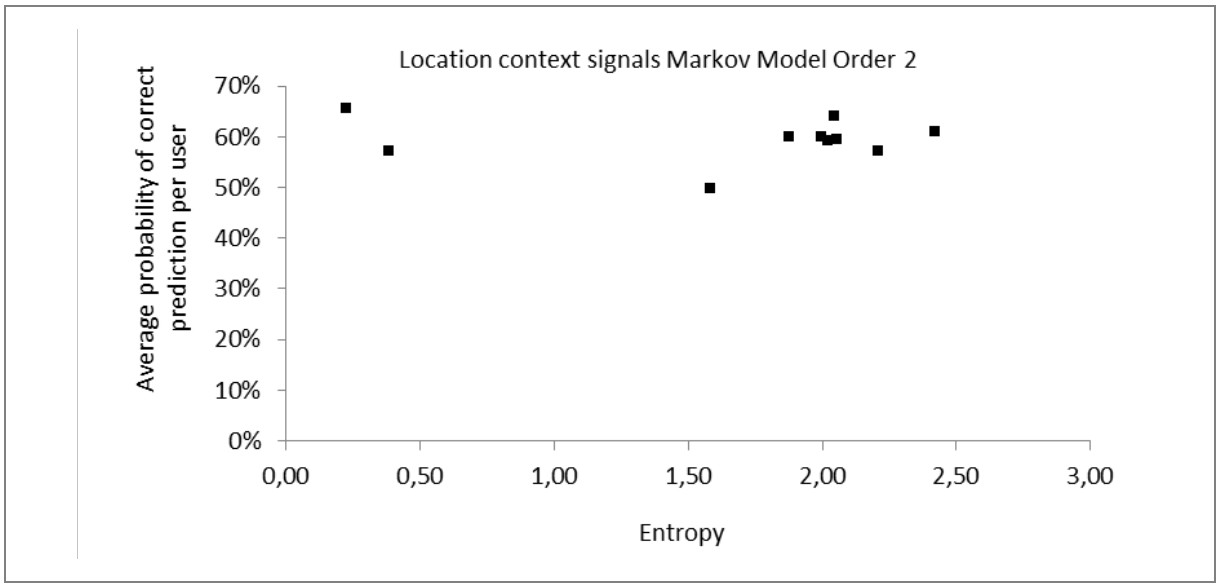

Figure 98 Correlating Markov Model Order 2 prediction for location datasets with entropy

When correlating Markov Model Order 1 and Order 2 prediction's on the location datasets with the entropy, the highest value of correct probability is achieved when there is a low entropy level (below 0.5) which corresponds to a high instance/symbol ratio (above 30) on all cases. This result implies that the less randomness there is in usage pattern, as shown by a low entropy value, and the more an application usage is repeated, which is shown with a high value of the instance/symbol ratio, there are more patterns that can be recognized by the algorithm.

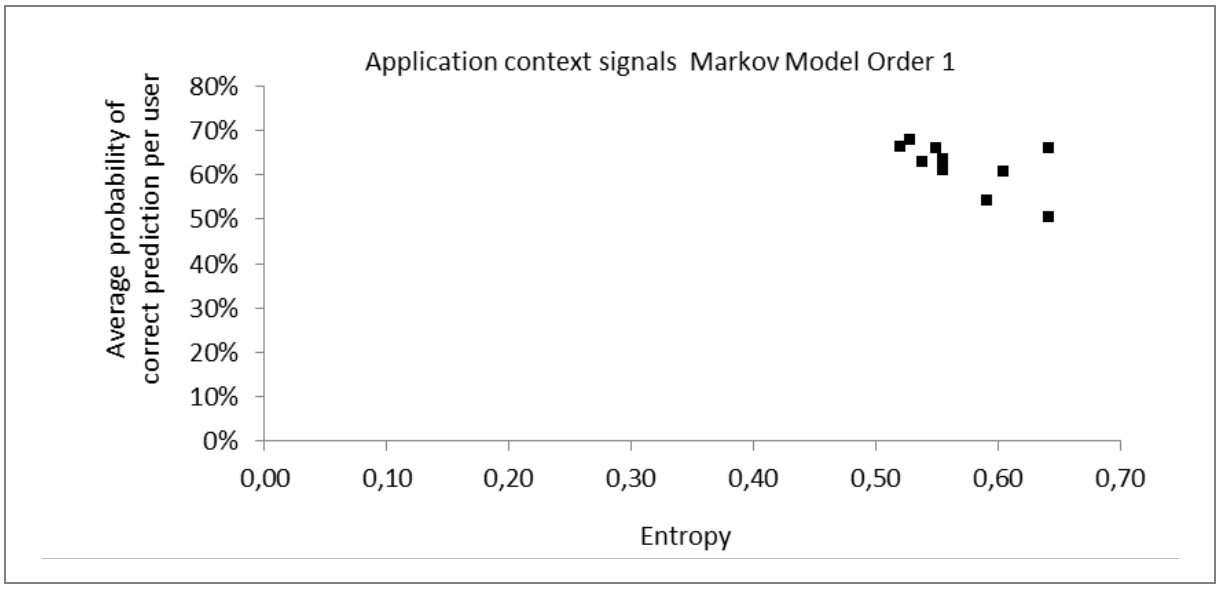

Figure 99 Correlating Markov Model Order 1 prediction for application datasets with entropy

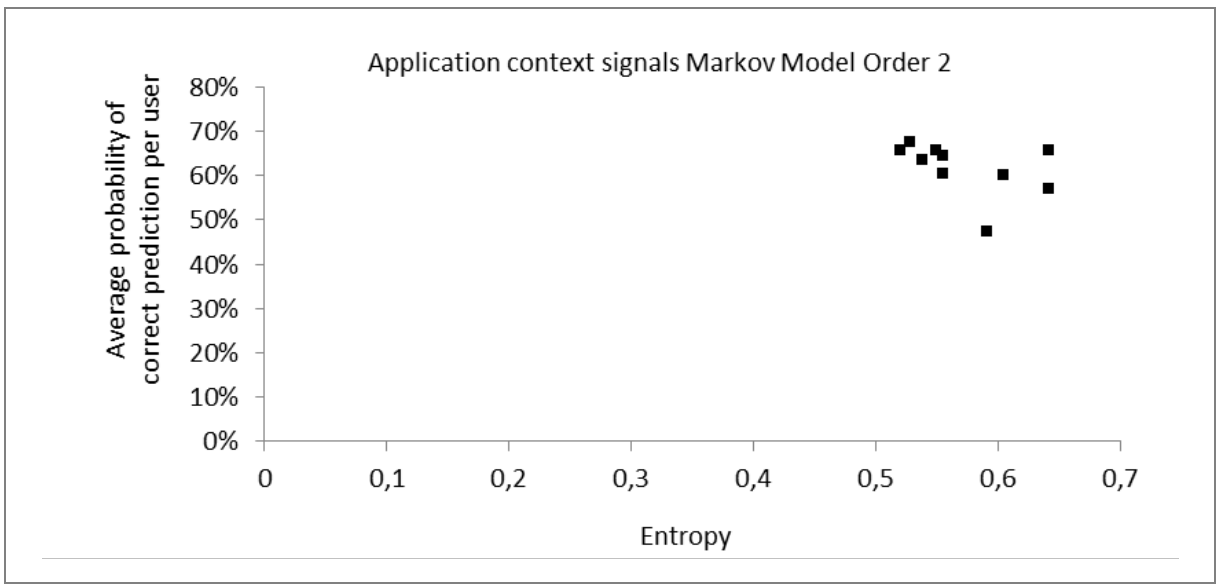

Figure 100 Correlating Markov Model Order 1 prediction for application datasets with entropy

When correlating Markov Model Order 1 and Order 2 prediction's on the application datasets with the entropy, we see that the lowest entropy values of c. 0.5 correspond to the highest prediction rates (66% to 68%). The highest value of correct probability for Order 2 corresponds to one of the highest instances/symbols ratio achieved (657), which again implies that there is a correlation with the randomness in behavior and the amount of times a pattern is repeated by the user that leads to a better prediction rate.

6.2.3 Improving performance

In the previous section we saw that the average prediction rates for both location and application context signals are below 80% in all cases, changing for each user, despite the

fact that there are all users from a similar demographic group and working in the same location. The differences and low prediction rates are due to the fact that there are a number of external parameters that vary widely from user to user that make predictability to vary as well (Chaoming Song, Zehui Qu et al. 2010). With the goal of personalizing the algorithm to the user's mobility patterns, we propose to include time in our development of the Markov Model and analyze the improvement it poses to use time with the context variables. Our contexts now consist of the context variable group and the time stamp. Our algorithm will read the symbol in the sequence, check its timeframe, and look up in the string of symbols that belong to the same time frame to make the prediction of the next symbol. We will now have different historic symbol strings, filtered by time frame. If there are more patterns associated with specific times of the day and thereby improve the prediction rate.

Methodology

We filter the variables by time by partitioning the day into different slots defined in our context model in Figure 14 in "Section 4.2.1 Main context variables": morning (7-12), afternoon (12-18), and evening (18-23); we disregard the night because there is less activity. We set these fixed time frames, and then we adapt them to each user's patterns as the prediction process continues.

In a second step, we adapt the time-frame to each user, since each person has a different mobile usage pattern. We do this by starting with a standard time frame, in our case we choose the time frame that has shown best results in our tests, and then we adapt it dynamically to each user's pattern, as we can see in Figure 101. We include this step in our implementation of the Markov Model algorithm, on which we filter the datasets by time frames. It first uses a fixed morning time frame, since we have seen that those are the times when the users are more active with the phone, and it then adapts it to the user's patterns.

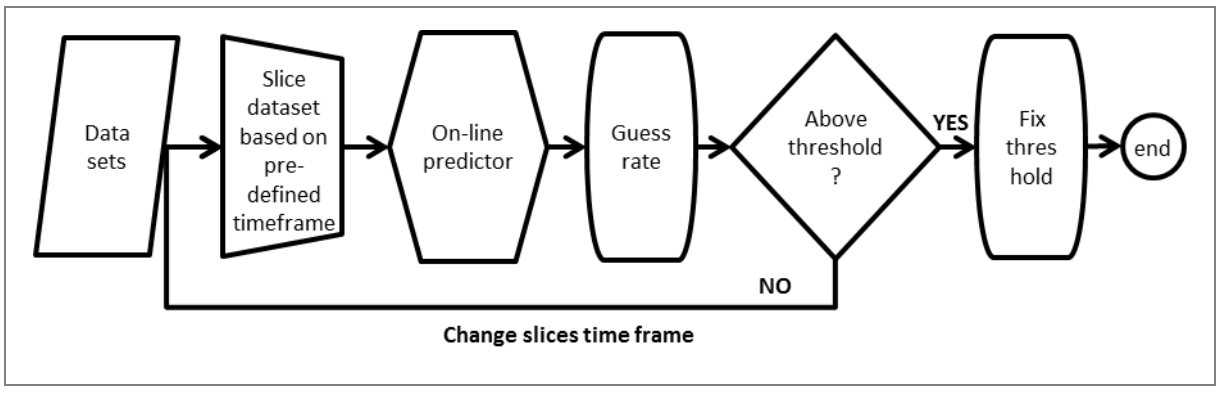

Figure 101 Adding time to the algorithm filtering the datasets

We repeat the tests of "Section 6.2.2 Performance results per user" for Order 1 and report the results for location and application context signals when adding time by the user's time frames patterns. We have chosen only to test Order 1 since we have seen earlier that

although Order 2 outperforms Order 1 for location context signals, this improvement is below 10% and it is not the case of application context signals (see Figure 85), where Order 1 outperforms Order 2 on six users. Moreover, Order 2 has worse prediction results when building the Markov Model and it also uses a larger amount of nodes when building the tree (Figure 86 and Figure 87).

Algorithm results adding time with a fixed time frame

Figure 102 and Figure 103 show the improvement in performance when filtering by the fixed time frames.

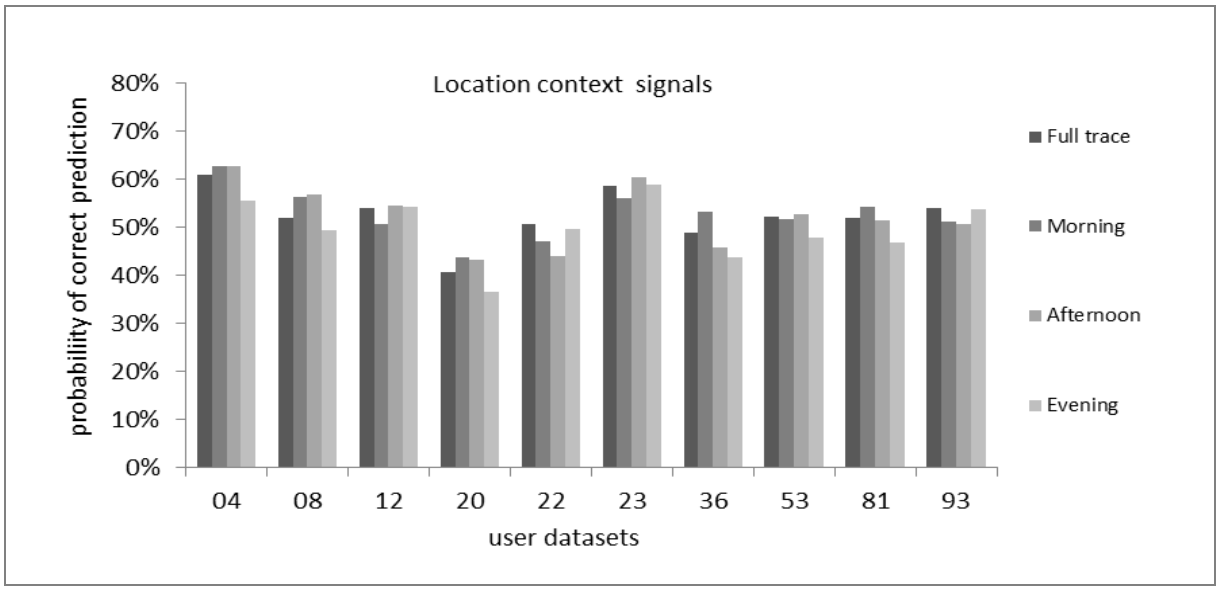

Figure 102 Markov Model prediction rate for location context variables adding a fixed time frame

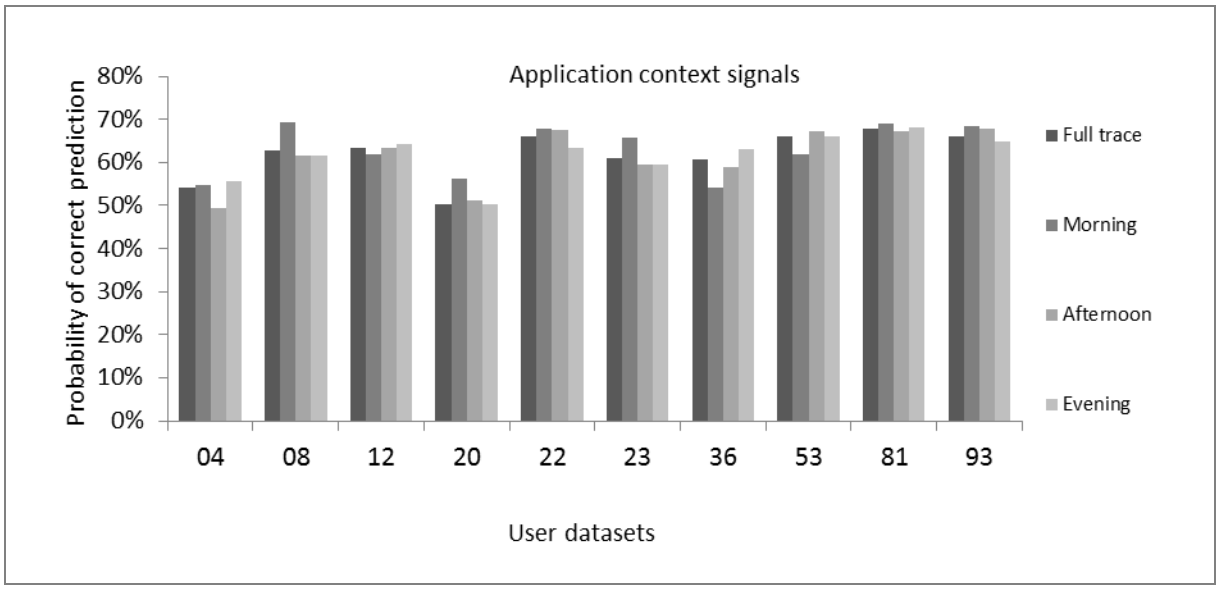

Figure 103 Markov Model prediction rate for application context variables adding a fixed time frame

We obtain very similar improvements in both context variables, an average of 1%. In all cases it is the morning time frame that shows the greatest improvement. In the location context variable case this is due to the fact that the subjects of the study have morning routines that are going to work to the same place daily and in the usage context because they all report to start their work schedule between 10 and 12, so their usage of the phone at that time will be higher.

Algorithm results adding time with an adapted time frame

We now test the algorithm adapting the user's patterns and restrict it to the time frames on which the user has a heavier and more repetitive usage of the mobile device, with the goal of further improving the prediction rates and also to reduce the number of nodes needed to build the tree by only needing one shorter instance history trace. This will imply that the prediction will only be done at this time-frame of the day. Table 28 shows the adapted time-frames to user's time of day on which they have more pattern repetitions.

User dataset	4	8	12	20	22	23	36	53	81	93
Adapted time frames	8-13	12-17	12-16	9-13	11-16	12-16	8-12	8-12	11-15	11-16

Table 28 Time frames adapted to users' patterns

Figure 104 and Figure 105 show the improvement in performance when filtering by the user's pattern time frames.

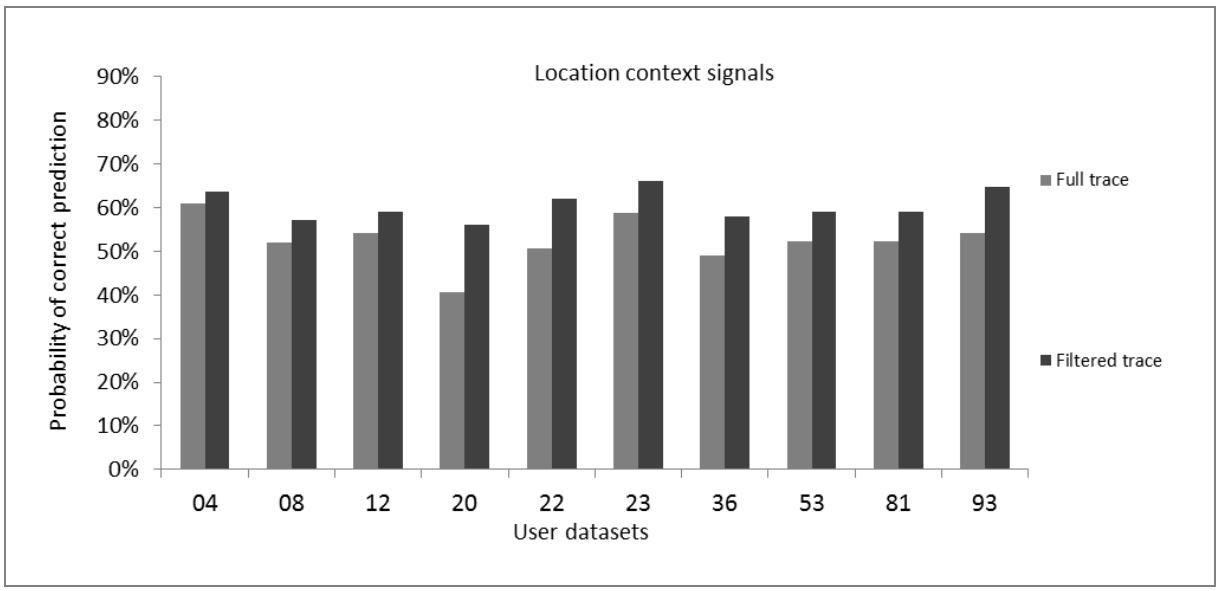

Figure 104 Markov Model prediction rate for location context variables adding an adapted time frame

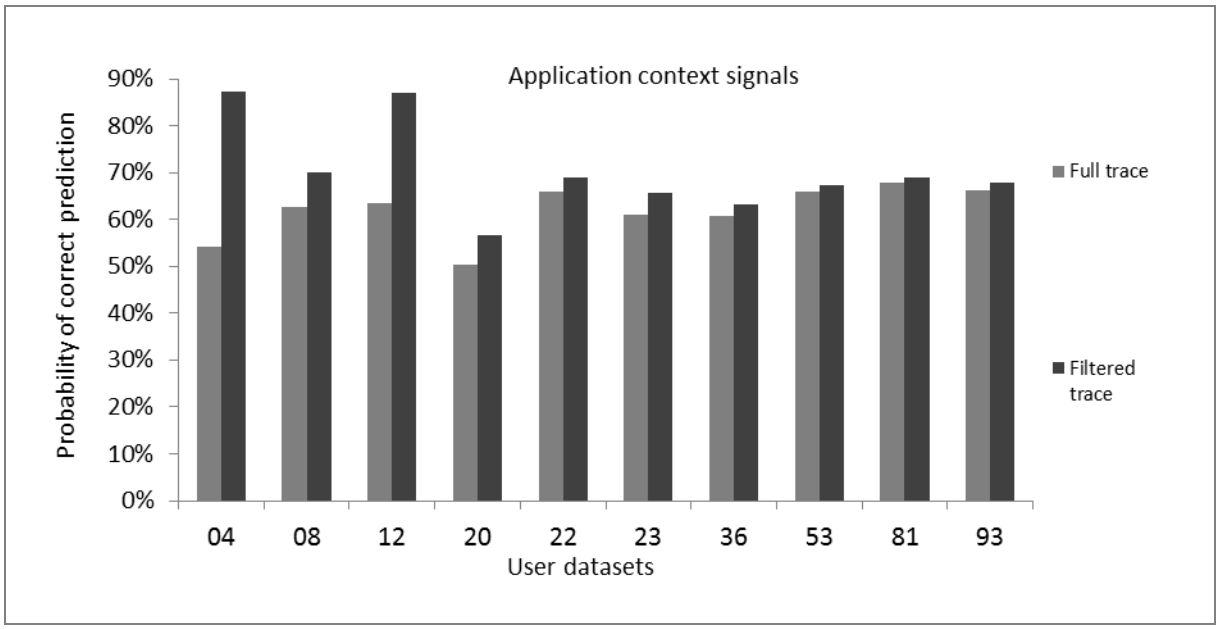

Figure 105 Markov Model prediction rate for application context variables adding an adapted time frame

When adding an adapted time frame, the results show an improvement of 11% in average, for the location traces, with no significant larger amount in any of the datasets. The prediction rates are still low, which suggests that there is still a high density of cell towers.

In the application context signal case, the results show that the probability of correct prediction improves by an average of 14%, and only on two users' datasets (04 and 12) the probability of correct prediction reaches a value close to 90%. These two users have in common that they reported a similar work schedule (both from 11 to 8pm), and somewhat regular patterns. This was confirmed by low entropy values in application uses of 0.5 in average (Figure 92) that affects their phone usage pattern.

Figure 106 and Figure 107 show the change in the number of nodes the algorithm uses to predict the next symbol.

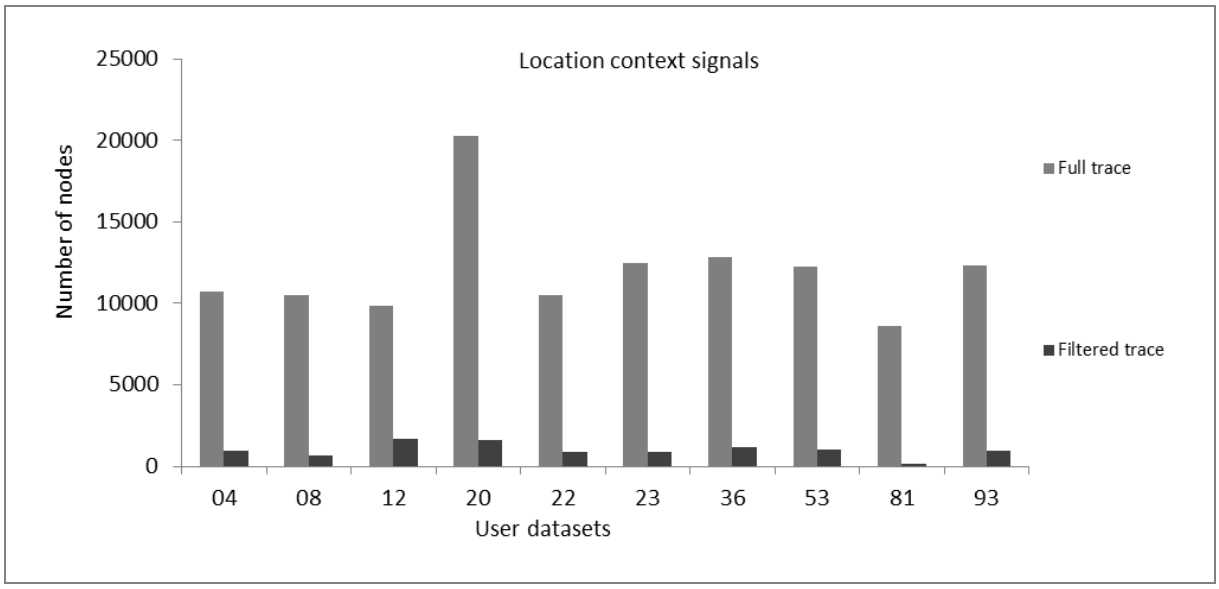

Figure 106 Markov Model number of nodes for location context variables adding an adapted time frame

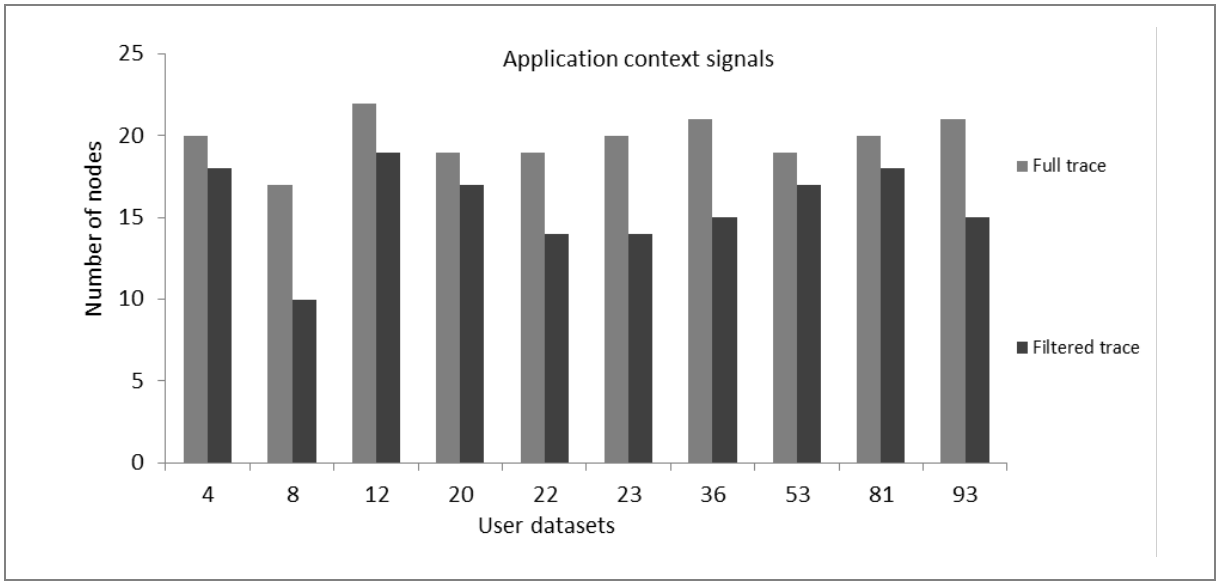

Figure 107 Markov Model number of nodes for application context variables adding an adapted time frame

When filtering the datasets adapting it to the user's time frame patterns, the number of nodes reduction is smaller in the case of application usage, decreasing by 23% only versus 35% of the location symbols. With the filter there are a limited number of applications combinations in use the phone in the chosen dataset.

Figure 108 and Figure 109 show the new number of instances the datasets have when adding time to the algorithm adapting it to user's time frames.

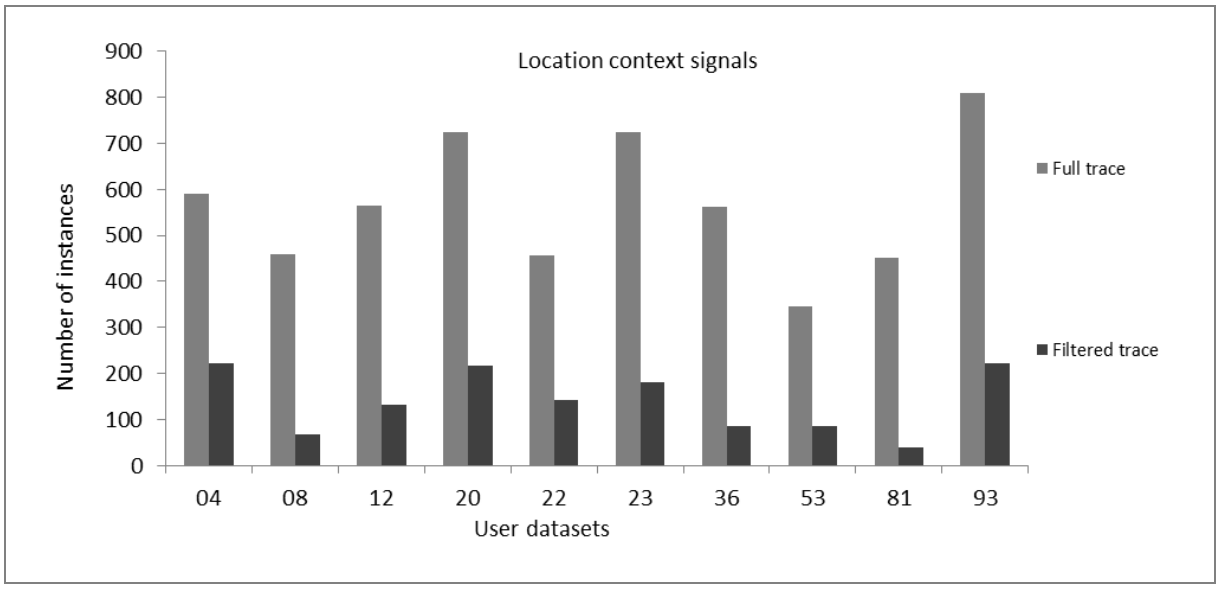

Figure 108 Markov Model number of instances for location context variables adding an adapted time frame

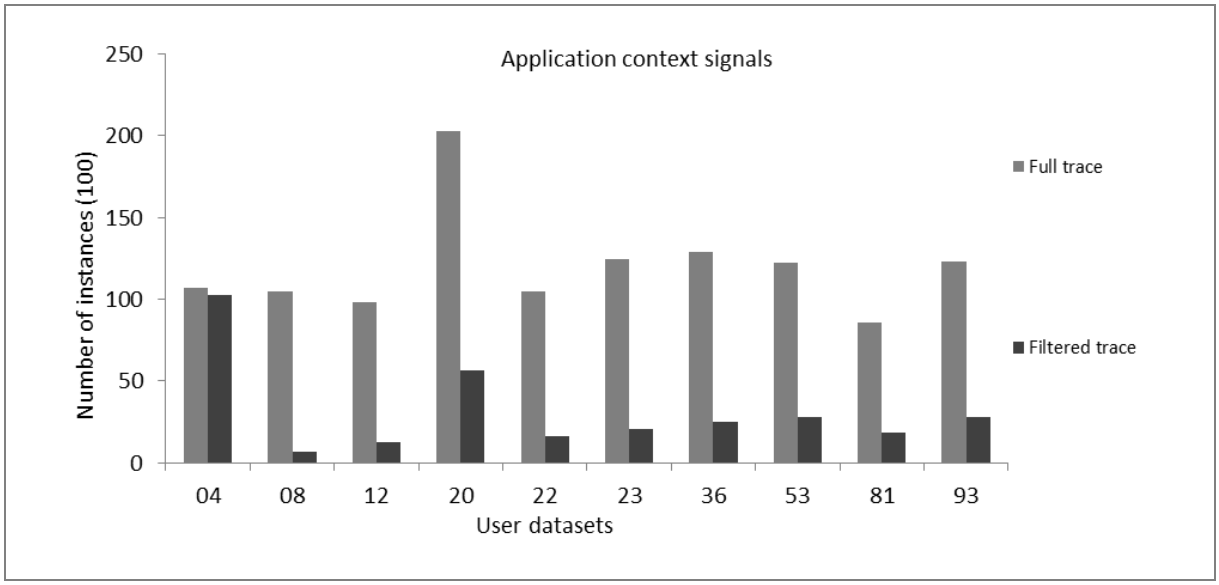

Figure 109 Markov Model number of instances for application context variables adding an adapted time frame

The amount of instances used is reduced significantly of the original size of the dataset in average for all users: 20% for the location context signals and 26% for the application context signals. Again there is an exception on user 04, where the amount of instances remains very similar (10728 versus 10200) which indicates that the phone is used heavily regardless of the time frames. We analyze the dataset of this user and confirm that when the phone is on it is actively being used.

6.3 How to include prediction in our model

In our tests we have seen that the probability of correct prediction is low, hardly reaching 70% in average with the time-frame improvement. These results are due to the existing limits of predictability in human dynamics (Chaoming Song, Zehui Qu et al. 2010) that restrain our capacity to predict user's behavior when using mobile devices, since this is prone to change radically at any given point.

We also know that these results do not improve when the amount of instances is larger, since it is related to the fact that at any point of time, a new symbol that has not previously been seen can appear due to the change of behavior of the user. This implies that we do not need a large amount of instances to achieve a good prediction rate, it can happen as early as after having read 1000 instances (c. 5 days of context signals). Therefore these algorithms could be stored in the mobile device since they do not need to keep a large amount of data to improve their results, which again would address our storage restrictions described in "Section 3.1.4 Challenges".

In order to be able to delimit the prediction rate, we propose to limit the number of a-priori values the context signals can have to a maximum of 10 by grouping the values, following the same scheme we defined in our sensor data normalization scheme in "Section 4.3.3 Architecture phases" (in Figure 23). For the location context signal example, we propose to group the location data to relevant places in the user's life, such as home, work or elsewhere, which we consider to be the three basic locations and flexible enough to cover all the scenarios of regular life. The mapping would be performed following some basic classification rules similar to those we have used in "Section 4.4.5 Reasoning context (Phase 1)" and in "Section 5.1 Test framework" to classify our instances, on "Section 8.3 Future work" we will further explain how we will approach this.

We would derive the locations and confirm with the user that they correspond to our mapping. This implies that we will need to include a confirmation step here on which the user will need to confirm the location mapping done. This step will also help to eliminate possible privacy issues that along with security and data protection, is one of the major concerns for context-aware applications since it can be considered by users as intruding (Korpipää and Mäntyjärvi 2003), several strategies are under discussion, such as privacy by design (i.e., requiring intervention and explicit acceptance from the user), privacy by law (i.e., defining the sphere of personal data not subject to use by applications) or user empowerment (i.e., keeping users in complete control of their personal data, able to switch on and off the application and data).

From an end user point of view, the application would work as a regular application but it would include an auto-training phase and a user confirmation of some data while training. Figure 110 explains the steps to get the application to run.

1. User downloads the contextualization application into the phone
2. The application displays a message explaining how it works,
    - Specifying that it will use context to adapt the application to the user's behavior
    - Explaining that the information retrieved from the user will be transformed and encrypted to avoid privacy issues
    - Stating that it will require the user to confirm some information for the application to learn from the basic characteristics inherent to the user
3. The application runs in the background compiling the context signals of the user and sending to an external storage place (server or cloud) until it reaches the threshold that has been defined. During that period, of time, the application does not make use of context information.
4. Once the contextualization module has compiled the required amount of information that will allow it to obtain a good accuracy results, the SVM classifier is trained.
5. When the SVM has built is model, the contextualization module starts working and the application is able to make use of context states on its functionality.

The context module would "re-train" itself after a specific period of time.

Figure 110 Steps to run the contextualization application

The contextualization module is ready to classify with the current model we have tested (as specified in "Section 5.4 How to use classifiers in our model") although we recommend to re-train itself every 6 to 9 months, in December and July, to align itself with seasonality disruption in usage patterns. The prediction feature will need to compile context signals during an average of 5 days (approximately 1000 instances as seen on this chapter) to start producing acceptable results when guessing the future symbol. When testing the context signal prediction in our contextualization system ("Section 4.4.7 Predicting context"), we realize that many times it fails to predict the right context symbol, which is expected due to the variability of predictability in user's behavior. (Chaoming Song, Zehui Qu et al. 2010) that we have discussed on this chapter. These failures will lead to an incorrect context signals being passed to the classifier in our model, increasing the possibilities that it produces an incorrect context inference.

From a back-end point of view, the data flow is described in Figure 111, in which we conceptually explain how context model is built and how it would run.

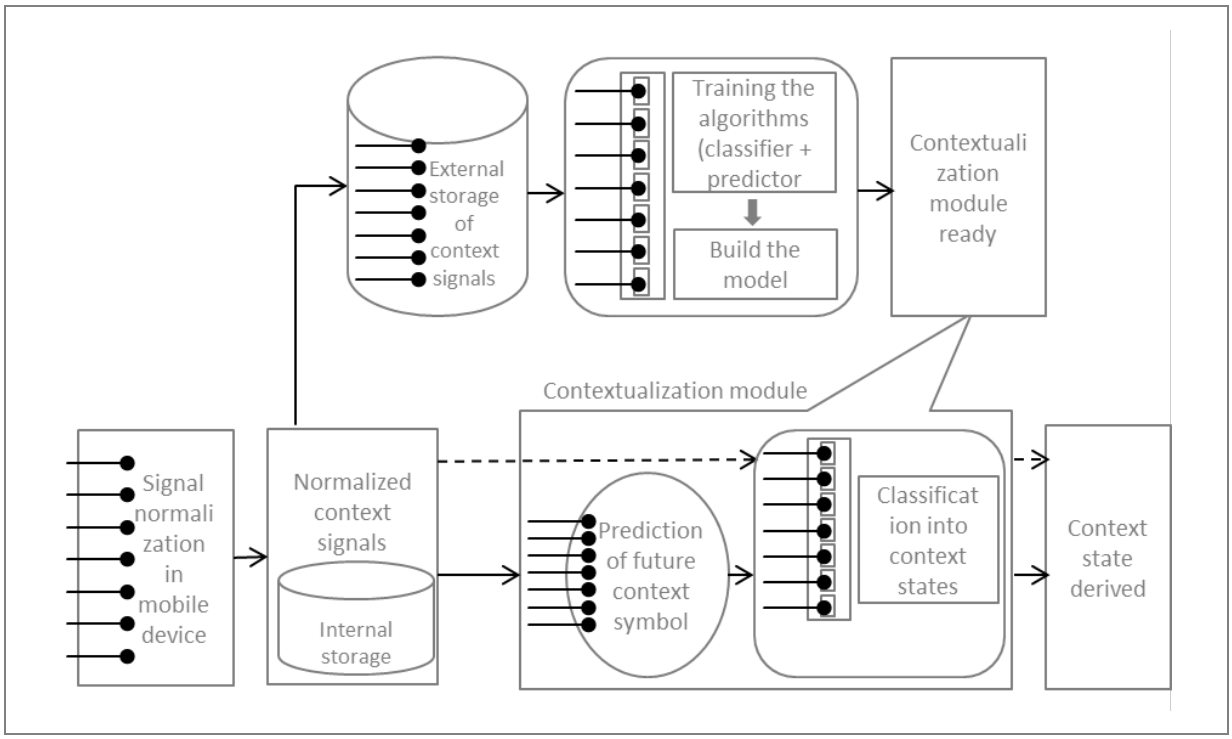

Figure 111 Context prediction model conceptual data flow

The context signals capturing and transformation would take place on the mobile device, while we propose to store the historic information and the creation of the contextualization model outside the mobile device. Although the data from the user is transmitted from the mobile device to an external storage for the training phase, it has previously been encoded into binary values, avoiding potential privacy issues.

Once the training phase is over and the model is ready, it would update the contextualization module installed in the mobile device. For storage purposes, once the contextualization module starts to function, it should periodically prune itself from old values so that it does not saturate the mobile device. By pruning the datasets we will be avoiding being overloaded with data that can lead us to build over-complex models (Domingos 2012).

We propose that the training phase should be done gradually, i.e. the first model would be created after the application has captured three months, and after that it would update itself by-weekly up to the point when it has compiled six months of mobile data usage. We then propose to retrain it again every nine months and update the model. The prediction functionality would start after compiling five days of mobile usage, which is the amount of time needed to compile the necessary amount of instances to allow the algorithms to produce a good prediction output, as we have seen in “Section 6.2.3 Improving performance”. In our model, we propose that our contextualization module can either transform the signals along the way by predicting the upcoming one or not. This means that the contextualization module can produce a future context state and/or a current one. We propose this functionality to provide flexibility to our model so that it will allow the

application to choose which context state to use. In the next chapter, we explain how applications can choose to contextualize results in a different way through our example of a mobile search engine.

6.4 Conclusions and discussion

In this chapter, we illustrated how to apply a context-awareness model for mobile devices with phase 3 architecture for context inference. We first explained how to model the context data using symbols that can be read by machine learning programs based on the MavHome project. We then ran a batch of tests for an in-depth analysis of the performance of the machine learning algorithms using real-life mobile context data that evaluated both their performance rate and memory consumption. We ran both Order 1 and 2 Markov Models, which use different window sizes for the previously seen symbols. We also proposed a performance improvement to our context mode by adding a filter to our symbols based on the time variable.

We have analyzed how to predict context states using a history of events in a mobile environment using the location and application usage as examples. Our goal was to gain an in-depth understand of the nature of mobile context data and test how well basic machine learning algorithms, such as Markov models, work for context prediction. The large set of user mobility data collected by the Reality Mining project has allowed us to experimentally evaluate the predictors, which provides insight into their performance. We have proposed a means to improve the performance of the algorithm by including the time variable with the other context variables, which allows us to more quickly obtain the desired prediction rate.

Our results have shown that we are at a stage where we can predict context using existing tools, i.e., contextual mobile data provided by mobile phones and current machine learning algorithms. Nevertheless, these algorithms have some downsides for use in mobile device search applications; one is that the context was derived as a simple single context variable rather than as a combination of context variables that would allow for a more sophisticated context state and support a more powerful context concept. In the future, we will explore using classifier algorithms currently in use for mobile searches that are more suitable for complex context derivation and run faster, which would provide more immediate prediction results.

When the instance size is 70% or less of the total signals, the predictor has 74% probability of correct prediction for location context signals and 84% for application context signals. The overall probability of correct prediction of the individual datasets is low on both algorithms, never reaching 70%, which indicates that there is a lot of randomness in the behavior of the users. Changing the order of Markov Model from 1 to 2 improves the prediction success rates always for the location context signals, this is not so evident for the application context signals, which means that when there is a largest amount of different

symbols Order 2 outperform Order 1 prediction results. The downside of increasing the order is that with Order 2, the algorithm more than doubles the number of nodes it uses when predicting the next symbol.

When we filter the Markov Model by time frame, we find a significant improvement of 12% in average in the prediction rate, but the results are still below 90%, except for some datasets of the application context signals. This is due to the amount of cell towers available in the topology of the landscape of the project that makes the delimitation of the number of symbols impossible, even when there is a specific time frame in place.

The analysis of the ratio between the number of instances and the number of symbols combined with the entropy level of each dataset, allows us to understand how the nature of the user affects the prediction results. So for example, the lowest entropy values of c. 0.5 correspond to the best prediction rates and the highest instances/symbols ratio.

The prediction functionality we add to the classifiers is suited for mobile devices when the number of instances used is kept between 2000 to 5000 average, which as we have seen, is a relevant amount to evaluate the accuracy rate of the algorithm.

In the next chapter, we provide an example of how to use contextual information to improve mobile device usage using a mobile search engine as an example. We explain how making the mobile search engine context-aware improves its understanding of the user's intent.

7 Mobile search: a real life example

The fast adoption of mobile access to the internet is leading to the prediction from some of the main industry specialists that mobile phones will soon be the most common method of web access worldwide. It has been shown that devices used for search are generally more sophisticated with rich features and high-resolutions color screens with XHTML support (Church and Smith 2007). Search engines, which are the main gateways for more than half of the users connecting to the internet, (Gómez-Barroso, Compañó et al. 2010), and one of the top 5 web sites visited over a mobile phones (Nielsen 2009), will need to adapt to mobile environments. Mobile searches will have to take into consideration the nature of handsets, which are pervasive and personal-centric and continuously capture information related to the user. If used with care, this information can be included to provide more meaningful search results by augmenting searches with real world information related to users' profiles and behavioral patterns. Context awareness can enhance the mobile search experience by augmenting user queries with context information captured through the handset's sensors. In particular the information can be used to better understand the user's search intent and improve the interaction effectiveness, which can improve the general goal behind searching in general, i.e. to provide quality search results efficiently (Brin and Page 1998) and adapt the results to user's information needs (Micarell, Gasparetti et al. 2007). Quality of searching has been the key topic for the last decade (Brin and Page 1998), understanding the intent underlying users' queries can help improve user satisfaction (Ashkan, Clarke et al. 2009). We believe we can use smartphones' sensing capabilities to include contextual data which can help us better understand the user's query intent in mobile searches.

On this chapter we explain how the context-awareness model we have proposed can be applied to a real life scenario to improve mobility uses such as a mobile search. In our search test examples we use the Bing search engine, Microsoft's decision engine launched in 2009 (Notess June 8, 2009) .

7.1 Search engines basics

Search technology emerged in the 1960s, and it was with the popularization of the World Wide Web and of the internet that it became popular (Bloem, Thomas et al. 2009). Search technologies based on information retrieval (IR) inherently predicated on users searching for information, whereas a web search is often based on a navigational or transactional need (Broder 2002). A search is defined as an activity that includes a query to a search engine (Church and Smith 2007), which directly retrieves documents from an index of millions of

documents in a fraction of a second, whereas a query is a user request for information through a search engine's user interface (Micarell, Gasparetti et al. 2007).

Web searches can have different taxonomies based on the query intent:

1. Navigational: if the intent is to reach a specific site immediately.
2. Informational: if the intent is to acquire information on one or more web pages.
3. Transactional: if the intent is to perform an activity using the web.

Search engines are sites on the Web designed to find information stored on other sites. In Figure 112, we can see a conceptual flow diagram of a query on the web, based on the information retrieval model defined by Broder (Broder 2002).

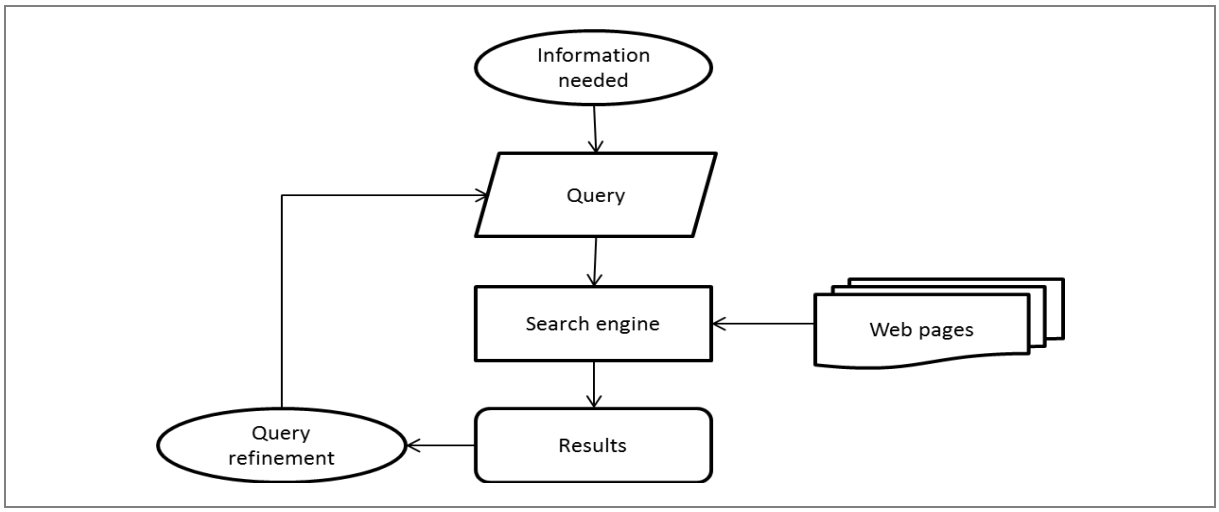

Figure 112 Information retrieval model for the Web

Search engines check the position at which the search keyword appears on pages and how frequently the keyword appears in relation to the other words on the page. If a page is relevant, the searched keyword will appear quickly and more frequently than the surrounding words. Search engines return a search engine results page (SERP)(Ashkan, Clarke et al. 2009), which is a list of pages that contain the queried keyword.

All search engines have three basics tasks in common (Franklin 2004):

- They search the Internet based on important keywords.
- They keep an index of the words found and their location.
- They allow users to search for words or combinations of words found in that index.

Search engines have evolved over the last decade (Broder 2002):

- First generation search engines (from 1995 to 1997) mainly targeted on-page data (e.g., text and formatting).

- Second generation search engines (from 1998 to 1999) relied on web-specific data, such as links or anchor-text, as their primary ranking factors.
- Third generation search engines (from 2000 to today) attempt to blend data from multiple sources to try understand the query intent, i.e., to answer ‘‘the need behind the query’’; they make use of semantics, dynamic data and context.

It is with these third generation search engines that smartphones appeared and enabled mobile search; smartphones are ideal platforms for search personalization because they provide user-related information that allows for adaptation with increased user interaction. Currently, only a few search engines possess tools that adapt to user interaction (Micarell, Gasparetti et al. 2007).

7.2 Mobile search

A mobile search is defined as the practice of querying a search engine from an Internet-connected handheld device, such as a smartphone (Church and Smith 2008) (Kolmonen 2008) (Krum 2011); the software designed for a mobile device that provides the means through which a user can submit a query and receive a list of results matching the search criteria (Kolmonen 2008).

Figure 113 is a simplified, conceptual diagram of the data flow during a mobile search.

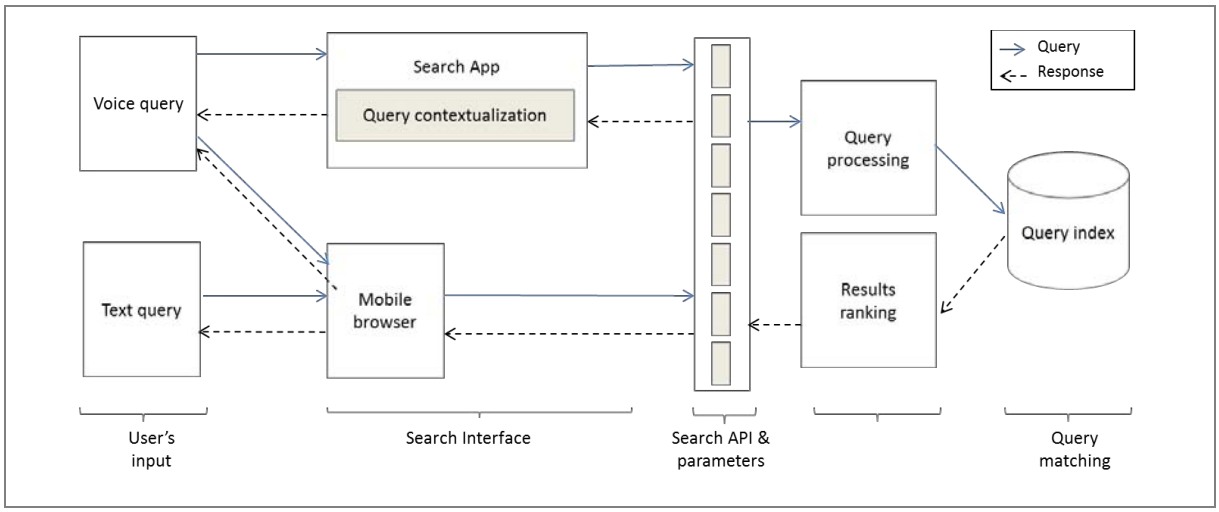

Figure 113 Mobile search engine conceptual data flow

The key components behind the mobile search engine are usually the same as in a fixed environment. However, there are some limitations in a mobile device, such as small-screen size, limited text-input capabilities and brevity of mobile internet connectivity, that limits retrieval opportunities (Kolmonen 2008), (Church and Smith 2008).

Mobile search today is typically carried out either via webpages of search engines displayed in a mobile browser or via dedicated search applications, often native to the device. Users

can input a query by text (typing on the device's keyboard) or by voice (speaking into the device's microphone). The query is captured through the search engine's user interface, which is either the mobile application or the mobile browser.

It is primarily the search application, as opposed to a standard web browser, that allows us to contextualize queries by gaining access to a mobile device's many sensors.

Today's commercial solutions for mobile searching do not offer significant, differentiated capabilities compared with those for traditional, desktop-based searching. Most major players opt to adapt their web interfaces and APIs to meet the mobile device characteristics (Church and Smith 2008). Mobile search queries possess specific characteristics, such as being shorter in length than their web counterparts, exclusion of advanced queries features like Boolean commands and the low probability that users will look beyond the 1st page of results (Church and Smith 2007), (Kamvar, Kellar et al. 2009). Thus, the mobile search nature should be taken into consideration. Over the last four years in the academic domain, several researchers have proposed specialized mobile search engine designs centered around three main concepts:

- Interaction paradigm: Some authors propose a new methodology to process the query and categorize the results, such as Yahoo's one-search (Yi, Maghoul et al. 2008).
- User interface: To address the problem of reduced screen real estate, many authors propose to categorize the results of the query in the display as in the Findex browser (Heimonen and Käki 2007) and Yahoo's one-search (Yi, Maghoul et al. 2008).
- Context usage: Other authors propose to combine context information with the query text and also include location, as in Yahoo and Google mobile searches, or other contextual cues, such time and social data (Church and Smith 2008).

With modern web standards such as HTML5, some of today's search interfaces are already able to capture certain types of context information, such as location and time.

7.3 Embedding context in searches

Personalization of search usually occurs during information retrieval or filtering (Micarell, Gasparetti et al. 2007) in one of three ways: as part of the retrieval process, re-ranking or query modification.

We propose a generic architecture for processing context signals and making them available to downstream applications, such as a search engine, by adding a contextualization module that is made of two components:

1. query contextualization
2. result contextualization to the mobile search data flow

When we apply context, we contextualize queries by inserting into the search engine the contextualization model that we have defined and by enhancing the queries with contextual data. We need to include a contextualization module in the search engine that makes queries context-aware. As observed in Figure 114, this module possesses a context-aware architecture, takes the query as an input, contextualizes it and produces the result as an output.

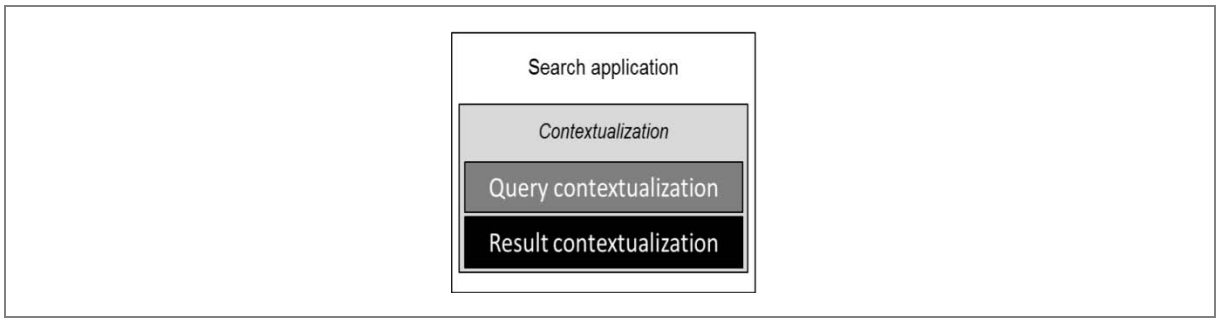

Figure 114 Contextualization module

Outside the contextualization module, the search application transforms the query text and various context signals into appropriate parameters for the search API. Once a request has been made to the search engine via its API, there will be a several internal query processing steps before the query is sent to the search engine's index to retrieve relevant documents. A key component of the search data flow is the search API itself because it defines the supported input and output parameters for a search query and its response. Apart from the query string, the APIs of most commercial search engines support a several parameters with either of two functions:

- define which types of documents should be returned by considering document language, number of documents, adult result filtering, search vertical (e.g., images, news, video and local) and whether to perform certain types of query processing.
- provide some additional context parameters for the query, such as market and location.

Context signals are processed in four phases; as explained in the proposed architecture in Figure 19, the first two phases are application-independent and solely concern the acquisition of context signals and the modeling of higher-level context states, whereas the last two phases make use of application-specific logic and input/output routines. We will only explain the last two phases here.

- **Phase 3: Application state updates**

  Inputs: Context states, user inputs

  Outputs: None (application state)

  Description: After having gained a detailed, robust understanding of the context, the application is able to update its internal state. Apart from the context, the application can at this stage also use explicit user inputs and other application state variables. For

example, to perform a search, we combine the user's query with context states to produce a contextualized search engine API query.

- **Phase 4: Contextualized output**

  Inputs: None (application state)

  Outputs: user-directed interface actions, communication with other processes/applications

  Description: Based on its updated state, the application can produce contextualized outputs, which can either be user-facing, as with the contextualized presentation of search results, or call external services in a contextualized fashion, as with calls from the augmented search engine API (both mentioned earlier).

Where to place the contextualization module is a discussion that depends on the search engine implementation for the mobile device, i.e., if it is simply a shell with all functionality running on the server side or if it can be implemented on the client-side with its own intelligence embedded on it like other personalization systems. Just like the personalization functionality, the contextualization module can be placed at three different places in the search process (Micarell, Gasparetti et al. 2007):

1. As part of the retrieval process and the ranking; placing the module here would be computationally consuming and slow the search process.
2. After the ranking occurs, including contextualization on the client side; placement here would also be time consuming and slow down the search.
3. Query modification and augmentation with contextual information outside the retrieval step; placement here would ensure that such modification does not alter the search process.

We opt to take the last approach and to modify the query either before the search if the query needs to include contextual information or after the search if the presentation needs to be adapted. To implement this mode, we and other authors believe that a new search browser interface should be created that integrates context information cues (Church and Smith 2008) captured by the mobile device and that processes them in the contextualization module.

7.4 Enhancing mobile search through context-awareness

Mobile search results differ from web results due to the nature of the mobile device. The content of search engine index pages differs by the terms that appear on the pages; there are fewer terms in mobile search due to the size, which limits the vocabulary (Church and Smith 2007). However, by adding contextual data in a mobile search, we can improve mobile searches by enriching the query with additional information that may facilitate its

interpretation. Some of the recent innovations in improving the mobile search interface are the result of using contextual information to enable predictive text input based on other people's inputs (Church and Smith 2008) and of localizing results based on the current position of the mobile device (Yi, Maghoul et al. 2008), (Heimonen and Käki 2007), (Church and Smith 2008). We hypothesize that including contextual data as part of the mobile search flow can improve the mobile search experience in two fundamental ways that can clarify query intent and improve search interface interactions:

1. Adapt the presentation and interaction to the user's context and environment; this involves adapting the user interface and interaction paradigms to the current context.
2. Clarify the user's intent, thereby improving the retrieval and ranking of relevant documents; this involves using the context signals to clarify the user's search intent and return more relevant results.

Table 29 lists examples of how richer contextual data could improve the utility of mobile search applications.

Context variable	Possible uses for search	Adapt output
Signal coverage level of the device	If signal is weak then reduce the richness of results to limit amount of data that needs to be transferred.	presentation results returned
Connectivity methods active currently	If WiFi is on then use it as location input and favor results that are relevant to the location. Use the network ID as an information source for possible activities as networks will generally allow us to distinguish between major environments such as work, home, airport, local coffee shop. Adapt interface features such as query suggestions to reflect previous search behavior in the same environment.	presentation results returned
Environment information	If the light level is low the device may be in the user's pocket and it may be more appropriate to return the query result through voice.	Results returned
Device's current location	Augment the query to favor results relevant for the location.	Results returned
Time and time frame of the activity	Adapt interface features such as query suggestions to reflect previous search behavior. Augment the query to bias the results towards certain scenarios (such as going out after working hours)	Presentation results returned
Internal device information	If input method is voice then output through voice.	Adapt presentation
Device's motion and angle	If we detect that the user is on the move, adapt the user interface so that results can be clearly seen and further queries can be carried out easily without having to type	Adapt presentation

Table 29 Examples of how each context variable group can affect a mobile search

Based on the methodology in Table 29, we distinguish between contextualizing the search API query and displaying the search results. Similar to our prototypical definition of a context model in the previous section, there are several ways to define an appropriate state update model for our search application. For this thesis, we implement a number of heuristically defined rules to illustrate the benefit of the context signals available for several search

scenarios. Although rules such as these may be practical and effective in some cases, a more automatic and learned approach may be more suitable in other applications, which we will explore in the future.

We list some uses in Table 30, together with the state updates and sample search engine outputs as produced for a contextualized and un-contextualized version of the query. We used Microsoft's Bing search engine for all queries.

User query	**Hotel**
Context signal	**State = abroad, location = London (Context state == working)**
State updates	**Add location string to query words "hote"l → "hotel London"**
Example output	

User query	**Restaurant**
Context signal	**State = abroad, location = London, Time = 11-13 (Context state == moving)**
State updates	**Based on the query time, activity and location add extra qualifiers to query keywords "restaurant" → "restaurant London lunch"**
Example output	

User query	**Marooned**
Context signal	**Application == music (Context state = relaxing)**
State updates	**Based on application used and disambiguating words "marooned" → "marooned music video"**
Example output	

Table 30 Example scenarios

One important consideration when making context-sensitive augmentations to a search query is ensuring that the original user intent is not being inadvertently altered. This point is particularly important when additional query terms are added to a search query. Because all

related logic must reside within the search application, the biggest challenge is how to make this decision without resorting to the internal knowledge maintained by the search engine about queries, such as historical click-through information and intent classification results. One potential client-side solution to this problem is using query similarity metrics based on the result sets alone. As described in research on query reformulation (Daume and Brill 2004), the similarity of two queries q and q' can be defined in terms of the similarity $s(r, r')$ of their according result sets r and r'. In our case, we can compute $s(r, r')$ by counting the number of matching *urls*, as well as sub-domains, appearing within r and r'. Based on this idea, the following scheme can be used to determine whether a contextualization is appropriate for any given query:

1. Generate contextualized and un-contextualized query versions q and q'
2. Obtain search result sets r and r' for q and q', respectively
3. If $s(r, r') < t$ for an appropriate threshold t then we conclude that the contextualization was appropriate and we accept result r, otherwise we fall back to the un-contextualized result r'

Often, the appropriate state updates can benefit from a deeper understanding of individual user characteristics. For example, Church et al. highlight diverse user characteristics exhibited during their daily study (Church and Smith 2009) (Church and Smith 2007) related to mobile search behavior. Therefore, individual query habits and history should ideally be considered when contextualizing a search application, especially when targeting a platform as user-centric as a mobile phone. In this sense, the context information can be considered as an input to a more general inference engine. Storey et al. give a representative example of such an inference engine that aims to model user profiles, but not other contextual information, for query reformulation (Storey, Sugumaran et al. 2004). Additionally, the time of day "impacted the popularity of queries as well as the nature of users' search topics" (Kamvar, Kellar et al. 2009); therefore, including time as a contextual variable in queries can also clarify the query intent.

7.5 Conclusions and discussion

Mobile search is currently a developing process, but there are several opportunities for improvement based on the existing limitations in using a smartphone's interfaces as input devices and on the lack of mobile content, which makes indexing and thus mobile search challenging (Church and Smith 2007). To engage mobile users, the search experience and quality of search results must improve (Church and Smith 2008). Contextualizing mobile search engines ameliorates these issues by augmenting queries and clarifying the intent of each query.

When embedding context into queries, we must also account for the potential risks associated with personalization of the search engine due to user discomfort with sharing personal information and to possible changes in the user interface that may disorient users (Micarell, Gasparetti et al. 2007). We ensure that only information related to the device itself, its usage or its location is used, and we do not include any personal user information. Additionally, we avoid altering the original user intent to preserve the uniformity and level of complexity of the user experience. Finally, we do not slow down the search process once an architecture approach is chosen in which the query is modified either before or after entering the search engine, and we recommend that the fastest classifiers be used to predict context.

We are currently determining how best to implement a pilot of the proposed context-aware, mobile architecture and subsequently test it in a commercial search engine. In our prototype, we will include the contextual class label “elsewhere” captured by the mobile device to enhance queries related to food and restaurants. We believe that by contextualizing queries related to these domains, we will achieve a more accurate understanding of the user’s intent that is needed to find a restaurant at any time.

First, the impact of including the context signal in a regular search on the query result must be tested. Once we have determined that we obtain a better result by contextualizing the query, we can then implement our prototype. In our design, the contextualization process will be split between the mobile device and the search engine. We will implement phases 1–3 on the mobile device, and phases 3–4 on the search engine. Therefore, the search engine will need to possess a new contextualization functionality.

To test the prototype in real life, we will follow a similar approach to Bifet et al (Bifet, Castillo et al. 2005) and investigate the influence of our predefined context states on the rankings for several queries using the Bing search engine.

8 Conclusions and further research

The growth of smartphones over the last few years have made them one of the most ubiquitous communication devices, with sales reaching 472 million units and accounting for 31% of all mobile devices sales at the end of 2011 according to Gartner (Gartner 2012), and by 2015, approximately a quarter of the planet's current population will own a smartphone (Thomas 2011). The success of mobile computing is related to how well the system adapts to environmental changes (Adelstein, Gupta et al. 2005), which together with the increasing flexibility of mobile platforms for third-party developers and the rapid increase in smartphone ownership indicates that 40% of smartphone owners will "opt-in to context service providers that track their activities" (Thomas 2011).

The use of context in ubiquitous computing amplifies human activities with new services that can adapt to the circumstances in which they are used (Coutaz and Crowley 2005). Context awareness can enhance mobile apps by adapting them to their environment and improving their usability by augmenting it with contextual information captured through the handset's sensors.

In this thesis, we presented a context-aware model and architecture to support ubiquitous computing targeted at improving mobility through the use of context. We investigate using the myriad sensors available in modern smart phones to capture the contextual data surrounding the mobile devices, their users, and their environments. Signals about user context are particularly valuable in mobile Web searches. Augmenting user queries with context awareness can improve both the understanding of the user's intent and their interaction with the search interface. We define the steps for capturing, processing and understanding context information to derive context states that can be easily embedded into mobile applications with the goal of reducing the inherent complexity of contextual data due to its heterogeneous nature.

In this chapter, we summarize the primary conclusions from our research in section 8.1, review the lessons learned while creating the thesis in section 8.2 and propose future work in section 8.3.

8.1 Summary

We have enabled mobile devices to make use of contextual data using our proposed context-awareness system that will allow them to improve their mobility. We have validated our system's modules using real-life mobile contextual information about a user's location, phone usage, and communication behavior.

The most prominent findings of our thesis research are the following:

1. **A broad analysis of the state of the art and background of context awareness**

   We provide an in-depth overview of the current literature on context awareness in chapter 2 that includes existing approaches to data acquisition, modeling and processing. While reviewing these designs, we realized that they were all ad-hoc with restricted scalability and would need to state how to handle context changes over time or how new services could be implemented.

   We analyze the contextual data and the machine learning tools used throughout the thesis on Chapter 3. We evaluate three different real-life datasets available in the literature and focus on the 10 traces from the Reality Mining project that contain the largest amount of and most relevant information. We describe the machine learning algorithms used for context prediction: Markov models for on-line learning with Bayes, Tree, Instance-Based and Rules classifiers for offline analysis. We also describe the underlying machine learning concepts.

2. **A thorough definition of context awareness focusing on mobility**

   In chapter 2, we review the different context definitions in the academic literature that focus on different aspects of contextual awareness, such as its nature, how to model contextual data, and the interactions between the users and context.

   We leverage these existing definitions and the components of context-aware systems to propose our own definition focused on mobile devices. We first define the underlying context as any relevant information that can be used to characterize the situation of an entity (a mobile device in our case) interacting with a user. We define a context state as a mix of multiple context variables with different meanings that when combined can affect the mobile application.

3. **Defined a context aware framework model to combine heterogeneous sensor signals**

   We created a coherent context model that explains how to capture and process raw data and translate it into contextual information usable by an application.

   In this model, defined in Chapter 4, we group context data into three groups by its nature relating to interactions with the user: what is happening (physical and digital interactions with the device), where this is physically occurring (location), and when the action occurs (time, date). We split these contextual variables into phone-related, connection, location, environmental and position categories. We transform the context variables and unify them into a context signal vector, which is the output of our context model. By unifying the data into a homogenous format, we create a framework that mixes heterogeneous data because each sensor captures information of a different nature and format. We synchronize all of the variables with respect to the time and calculate how long each signal is active in a specific time frame. We define the time

frame field as being flexible, so that it can be changed by the programmer to adapt to the type of application being developed. We implement our model programming in python using real contextual variables from the Reality Mining project.

4. Proposal of a mobile architecture to contextualize mobile services

In chapter 4, we propose a mobile application architecture that uses context information to improve mobile services. We define the steps needed to contextualize an application, which includes how to capture contextual data, make it understandable, infer context from it and use it. Using the real-life contextual datasets available from the Reality Mining project, we explain how to implement this architecture phase by phase. We explain how to obtain the contextual information available both within the mobile device and externally through the ContextLog application, which captures data using the phone's sensors and stores it in logs. We then transform the context data in the log files into context vectors and apply the context blending framework defined in our context model. Once we have the context vector file, we explain how to infer the context using machine learning algorithms. We provide a few examples of contextual data, namely location, battery usage and app usage, to exemplify how to derive context states from the available contextual data.

When implementing our architecture, we consider the limitations inherent to mobile devices related to battery consumption, memory needs, processing power and continuous wireless connectivity by designing and coding all of our programs to be light-weight.

5. Analysis of machine learning approaches for predicting context

In chapters 5 and 6, we report the results from an empirical evaluation of machine learning accuracy in a mobility environment tested using real-life data. We perform experimental tests using the mobile data available from the Reality Mining project (Eagle, Pentland et al. 2009) to predict context states. In our tests, we model the context data in such a way that they can be fed into a machine learning algorithm to infer the context state based on the available data.

We propose two types of algorithms, on-line and classifiers, to predict the context states.

- The on-line algorithms use the history of previous context states to predict the next context state. We used Markov Models to infer location and application use context.
- The classifiers use several variables for one time point to predict the context state. We use four different types of classifiers (rule based, Bayes based, Instance based and linear function based) to infer context.

Overall, we find that two algorithms are the most appropriate for our model with an accuracy rate above 90% and above 80% when including noise in the dataset, these

algorithms are the Tree C4.5 and Support Vector Machine. We recommend using the SVM classifier since shows robustness and keeps up a good accuracy rate regardless of the amount of noise we add to the datasets.

6. **Provide a real-life example of using context-awareness in mobile search engines**

   In chapter 7, we presented an example of how to apply our proposed context-aware model and architecture to search engines to enhance queries. In our proposed implementation, we account for the limitations mobile devices bring to search engines in terms of screen size along with the limitations of the search engines themselves, which do not account for contextual information beyond location. We propose placing the contextualization module between the query and search API inside the mobile search engine data flow and review the steps of the context-awareness architecture, explaining how to adapt it to a mobile search scenario.

We have published the content of this thesis in two articles:

- **Article 1: Yndurain, E., D. Bernhardt, and C. Campo, Augmenting Mobile Search Engines to Leverage Context Awareness. IEEE Internet Computing, 2012. 16 no. 2(March-April 2012): p. 17-25.**

  In this article, we propose a model that captures heterogeneous context data from various mobile sensors and develop an application architecture supporting context-aware mobile searches using real context data from the Reality Mining project. We analyze the sensors available on smartphones to capture contextual data and explained how these signals can be used in a mobile Web search scenario. We explain how augmenting user queries with context awareness can improve both the search engine's understanding of user's intent and the user's interactions with the search interface.

- **Article 2: Yndurain, E., et al., Context-aware mobile applications: implications and challenges for a new industry. ITP Journal, 2010. 4 (Part 4): p. 9-20.**

  In this article, we provide an overview of mobile context-awareness applications and outline some future areas of interest. We review the primary examples of context-aware mobile applications (pilots or commercial systems) and analyze their underlying architectures and computational models to understand the primary challenges facing the industry in making this concept mainstream. Our primary theoretical contribution is to define the design principles for contextualizing mobile applications and analyze their various architectures while identifying the challenges confronting context-awareness and the needs of mobile devices to make context-awareness a reality.

8.2 Lessons learned

The primary conclusions we have drawn in our thesis are that, when contextualizing applications, it is crucial to start with an end-goal in mind because this dictates which context states will need to be defined and what contextual information is attached to them. Hence, the implementation should be from the top down to design ways to apply context awareness and bottom up when developing the application.

When designing the application, we first define the mobility scenarios in which the context will be applied and then define the context states (including the specific context variables that compose it) needed to identify and contextualize them. Armed with such insights and background knowledge for each particular problem, we begin the bottom-up process to identify the context variables needed, the sensors that capture it, the nature of the available data and which learning algorithm to apply.

Another important factor is that privacy issues must be considered when designing context-aware applications in particular if there are any private or personal data that must be used. We solved this issue by proposing to implement the context manipulation functionality of phase 1 in the mobile device and to ask for confirmation of data mapping into location, when developing this, we would need to balance the possible usability negative effect that such confirmation might cause.

8.3 Future work

Context-aware computing is becoming more common in mobile devices, and many developers now include context signals such as location and device position in their apps. We believe that the context-aware applications today are not making full use of the contextual signals for three main reasons: #1 there is no standard model unifying all of the context variables that can be captured by the sensors, #2 there is no unified supporting architecture, and #3 there is no specific orientation to the design of mobile platforms.

In this thesis, we proposed a model to improve mobility uses through the use of context-awareness, while accounting for these three issues, and have provided a real-life example of how to apply our model using a mobile search engine. We are currently working on setting up real-life tests on how to improve the mobile search experience by including context classes we inferred in the queries with the Bing search engine. Our goal is to evaluate the impact on the query result by including different contextual variables. Once we have evaluated the success of a query including contextual data, the next step would be to implement our context-awareness module in the mobile search engine to test its overall performance.

An additional important question that would be interesting to research is the placement of a context-aware module within the mobile search engine between the mobile device and search engine server. This approach may improve the context-awareness runtime performance, particularly when there is a lot of information, and it might be better to delegate the computational power to the server.

Another research topic that should be addressed in the development of the application is the techniques used to cluster the context symbols into a fixed amount. In order to do this, we would follow current research on this topic for state aggregation (Satinder P. Singh, Tommi Jaakkola et al. 1995) and on (Chaoming Song, Zehui Qu et al. 2010) and analyze other existing approaches to clustering data.

As next step, the proposed architecture and contextual data model tested on this dissertation should be implemented, using a different OS and a more modern smartphone with new sensing features embedded within it. We performed our tests based on the Symbian OS, which was one of the few platforms that allowed for easy third-party application integration, enabled the installation of the ContextLog app for capturing contextual data, and had the largest market share of smartphones at the time when the thesis began and the data were captured. The laptop we used to implement the context-aware reasoning and test the machine learning algorithms had technical characteristics similar to those of a smartphone in terms of processing power. We conducted our tests to capture information from the sensors and store it preformatted for easier use by the context-reasoning engine. We should implement the first two phases of our architecture in a mobile platform that allows access to sensors APIs, the best OS at this point of time to do this would be Android due to its openness. In order to implement it in other two main smartphone's platforms, iOS and WP, we would need to get permission from Apple and Microsoft to capture all the sensor information that our model proposes in order to capture contextual data. In our development, we should test two placing the classification module on the mobile device and on the cloud to analyze the impact on its runtime as well as on the smartphone's performance. Ideally, when an application is not internet based, it would be preferable to have the classifier running over the phone, otherwise, it should run in the cloud (or in a server) which will allow it to have a richer amount of contextual data since it can store more information.

Appendix A: datasets evaluation

On this section we review and analyze three datasets focused on mobile usage that are available in real life studies in the academic world that can be shared by researchers with the goal of selecting the most appropriate one.

1. Reality Mining: sensing complex social systems. This project's objective is to analyze human relations based on their phone communications log.
2. Mobile communication and context dataset. This project's objective is to create a monitoring program that runs on mobile devices and captures context information.
3. Sensor signal dataset for exploring context recognition of mobile devices. This project's objective is to detect movement based in context variables.

On these projects, the goal is to capture contextual data related to mobile usage behavior during a period of time, with the objective to use it towards their context-aware research.

We analyze them and assess if they will be of use for our own context-awareness research, determining if the data they provide is good enough in terms of pattern repetition.

A.1 Evaluation framework

We propose a simple framework, based in two steps (overview and evaluation) to assess context data existing on each research project in a consistent way so that we can benchmark them.

Step 1: project analysis

In order to analyze each research project, we define an analysis framework in which we analyze three groups of information that will allow us to understand the project relevance for our research: what is the project about and who participates on it? What type of data does it capture and how good is its quality? How is data captured and what infrastructure supports it?

- Description: goal, use cases and users
- Data: timeframe and completeness, type of information captured and data sample.
- Infrastructure: monitoring software, device where the software runs and sensors used to capture information

Step 2: project evaluation

We base our choice on a specific criteria related to the project such as the history length (the larger the better), the degree of interest for our own learning objectives (the more aligned to the context data we want to learn the better) and the data quality (if it's been checked and fixed or not). We will rank each variable with the following values: low (L) or high (H).

- History length: traces are a compilation of user activities' when interacting with the mobile device captured through logs that contain contextual data during a period of time. It is considered to be high if it is larger than 6 months.
- Interest: How relevant and aligned is the contextual data for our research, if we can use it for the learning algorithms then it is considered to be high, otherwise it is low.
- Data quality: how precise is the data captured? If the authors have performed a quality assurance then it is considered to be high, otherwise it is low.

A.2 Project: Reality Mining sensing complex social systems

The project is available at http://reality.media.mit.edu/ and the datasets information is stored in a SQL data base and a matlab file.

A.2.1 Project description

Goal

On their research, Nathan Eagle and Alex Pentland analyze social interaction and human behavior from mobile phone data (Eagle, Pentland et al. 2009). They collect data from mobile devices on a period of time of a school year.

The researchers claim that it is useful to learn from user's application routines because it can help to optimize the phone usage design. For example, if an application is used a lot, it can place it in a more prominent place. They also claim that there is a relation between location, proximity and time to infer friendship by analyzing patterns.

Use cases

Data was captured throughout the regular life of each of the study subjects. The users that participate on this research are a mix of MIT media lab and Sloan business school which have a software application that monitors information on their usage of the mobile device. The origin location was always the MIT buildings (media lab or Sloan business school). They divide their scenarios into on-campus and off campus proximity (considering the first as a work scenario).

Aside from the log capturing, users answered some questions that were captured on a survey consisting of 25 questions related to phone usage and behavior.

A.2.2 Infrastructure information

Sensors used to capture data

Sensors used are those available to the Nokia 6600 mobile device: communication module and phone usage. They capture data on cellular tower transition, Bluetooth device discovery scans and communication events.

Software used to monitor data

A monitoring software is created, the ContextLog application, which is installed on the users devices. The application captures data from phone usage: call logs (voice or data), Bluetooth devices in proximity, cell towers, application usages and phone status.

Device used to follow users

The chosen device in which to install the application was the Nokia 6600 smartphone. Data captured was kept on the smartphone's memory card that was periodically collected by the researchers. Those who had GPS (30 people) had this data dumped to a server.

A.2.3 Dataset information

Length

Their study consists of 94 Nokia 6600 smartphones that have a software application pre-installed (called ContextLog) that collects data from the device. The period of time when this study happens in an academic year from September 2004 until June 2005.

Information captured

Three datasets are captured (1) Phone Log, (2) Bluetooth and (3) Location.

1. Phone Log: related to communication information. It captures the following information,

   Time = Stamp of current and end time.

   Description= Type of communication that happened (packet data, short message or voice call)

   Direction = Where the communication is heading to (incoming, missed call or outgoing).

   Status = What happens with the communication (connected, delivered, disconnected, failed, no delivery, pending and sent).

   Number = who is the user communicating with.

   PhonenumberID = what is the phone number of the user who owns the phone.

PersonID = User's id, who is the owner of the phone.

Duration = How long did the communication last, measured in seconds.

2. Bluetooth: related to the nearby Bluetooth devices, scanned every 5 minutes. It captures the following information,

   Time: Stamp of current and end time

   Devices: Information relative to the device, identification of the device (ID), MAC address, person ID and name of the device.

3. Location: Information relative to the cell. It captures the following information,

   Time: Stamp of current and end time

   Cell Area: Cell's name and tower information.

   User's definition of the location name.

   Service provider: what carrier is providing connection.

Sample dataset

Figure 115 shows a sample of a dataset of the project.

```
Communication Log: (date, text/call, incoming/outgoing, duration, number)
20040720T211505 DESCRIPTION: Voice call DIRECTION: Outgoing DURATION:
23 NUMBER: 555-432-9999
Phone Status: (charging, idle/active, current application in use)
20040721T095311 Charger: 1
20040721T083501 UserActivity: idle
20040721T095541 ActiveApp: [100058b3] Phone
Proximate (Visible) Bluetooth Devices: (date, mac, [device name], ...)
20040721T111222 devices: 000e6d2a3564 [S11] 000e6d2b06ea [t610]
Celltower ID: (date, area, cell, network)
20040721T111642 area, cell, nw: 24127, 2421, AT&T Wirel
User-Defined Celltower Names: (area, cell, network, name)
24127, 111, AT&T Wirel My Office
24127, 182, AT&T Wirel My Apt
```

Figure 115 Dataset of project Reality Mining: sensing complex social systems

Although the project captures information through questionnaires, it also stores information acquired through the context log application, which is what we are interested in.

A. 3 Project: Mobile Communication and Context dataset

The project is available http://www.cs.helsinki.fi/group/context/latest/docs/ and the datasets information is stored in XML files, communication and context logs are in separate

files (context-<id>.xml and comm-<id>.xml). The structure of both files is described in log.dtd file.

A.3.1 Project description

Goal

On his research, Mika Raento analyzes logs of communication (calls and text messages) and other context variables (location, profile and calendar) for a small number of people (25) over a period of time, which is not specified (Raento 2004; Mika Raento, Antti Oulasvirta et al. 2005)

Use cases

Data was captured throughout the regular life of the subjects, daily life routes, work and leisure.

A.3.2 Infrastructure information

Sensors used to capture data

The application supports four sensor types, all part of the mobile device's features,

1. location, including Global System for Mobile Communications (GSM) cell identifier, cell-based semantic location, naming of cells via network location services, and GPS via a Bluetooth GPS receiver;
2. user interaction, including active application, idle/active status, phone alarm profile, charger status, and media capture;
3. communication behavior, including calls and call attempts, call recording, sent and received SMS, and SMS content;
4. physical environment, including surrounding Bluetooth devices, Bluetooth networking availability, and optical marker recognition (using the built-in camera).

Software used to monitor data

Data has been gathered through a software that runs in the background of the mobile devices, the ContextPhone, a software platform consisting of four interconnected modules that sense, process, store, and transfer context data. (Mika Raento, Antti Oulasvirta et al. 2005).

Device used to follow users

The device where the application runs must use Symbian OS, the chosen device is the Nokia 7650.

A.3.3 Dataset information

Length

The data consists of an anonymous mobile phone communication log (calls and text messages), location and profile changes as well as calendar events for a small number of people over periods varying from a couple of months to a year.

The software has not always been running and neither is the phone always on, hence there are some significant gaps on the data collected.

Information captured

The communication log lists calls and text messages including date and time, duration, direction and originator/recipient of communication. The same identifier is always used for the same phone number/contact. All data has been anonymized.

The context data consists of location data in the form of GSM cell information (network operator, location-area-code (LAC) and cell identifier (Cellid)). Specifically:

- Current GSM Cell Id, Bluetooth devices around it
- GPS data from a Bluetooth GPS receiver
- Phone profile, Phone idle/active time
- Charger status
- Incoming/Outgoing calls, Incoming SMS
- User interaction with the Phonebook and recent call log
- Media captured with the device (photos, audio, video, text)

In addition to the values the logs contain stop and start markers.

Sample dataset

We can see an example of the communication XML in Figure 116.

```
<events>
 <event>
  <datetime>20030213T173147</datetime>
  <communication>
   <start />
   <comm.sms />
   <comm.incoming />
   <comm.number>1</comm.number>
   <comm.contact_name>1</comm.contact_name>
  </communication>
 </event>
 <event>
  <datetime>20030213T182451</datetime>
  <communication>
   <comm.call />
   <comm.incoming />
   <comm.duration>216</comm.duration>
   <comm.number>2</comm.number>
   <comm.contact_name>2</comm.contact_name>
  </communication>
 </event>
```

Figure 116 Dataset of project Mobile Communication and Context Dataset

A.4 Project: Sensor signal dataset for exploring context recognition of mobile devices dataset

The project is available at http://www.cis.hut.fi/jhimberg/contextdata/index.shtml and the datasets information is stored in ASCII file format that contains raw data and processed context atoms.

A.4.1 Project description

Goal

On their research project, Jani Mäntyjärvi, Johan Himberg, Petri Kangas, Urpo Tuomela and Pertti Huuskonen explain how they record sensor signal dataset for exploring context recognition of mobile devices (Mäntyjärvi, Himberg et al. 2004). They describe a set of real life usage test scenarios, collection and pre-processing of data. The focus on monitoring user activity and environment, using a sensor-centric approach to investigate context recognition. They examine signals carefully to try to find patterns that correspond to real life usage patterns that are captured through an external sensor box.

Use cases

The authors define five user scenarios performed by two users. Each scenario is repeated about 25 times and they last about 2 to 5 minutes. When the terminal was not on the table

it was hanging in front, from the user's neck. The data was annotated with video recordings. One sample recording was collected form each of the scenario. Later on the video recordings were sliced into picture sequences and time was synchronized with sensor data enabling qualitative examination of data analysis. Scenarios specification can be seen on Table 31.

Scenario1	Location	Scenario2	Location	Scenario3	Location	Scenario4	Location	Scenario5	Location
start	office	Start	office	start	office	start	office	start	office
walking	corridor	walking	corridor	walking	corridor	walking	corridor	walking	corridor
walking	stairs	walking	stairs	halt	lift	Sitting + walking	meeting room	halt	lift
walking	lobby	Halt	mail lockers	walking	corridor	Walking + talking	corridor	walking	corridor
walking	street	walking	yard	halt	balcony	Sitting + talking	coffee room	halt	lift
walking	lobby	Halt	mail lockers	walking	corridor	walking	corridor	walking	corridor
walking	corridor	walking	stairs	halt	lift	stop	office	stop	office
walking	stairs	walking	corridor	walking	corridor				
stop	office	Stop	office	stop	office				

Table 31 Scenario specification

A.4.2 Infrastructure information

Sensors used to capture data

The sensors used are the following ones:

- Accelerometer x-, y-, z- axis (type ADXL202JQC): Measures accelerations of the device in orthogonal directions
- Illumination (type IPL10530D): Measures the level of the illumination in immediate environment of a device
- Thermometer (type TMP36F): Measure the level of the temperature in immediate environment of a device.
- Humidity sensor (HIH-3605-B): Measures the level of the air humidity in immediate environment of a device.
- Skin conductivity self-made sensor: Detects contact between a device and the user's hand.
- Microphone (customized): Measures audio from immediate environment of a device.

Software used to monitor data

In order to monitor data, a sensor box is built for examining context-awareness which can be attached into a mobile device. It has two-axes accelerometers inside connected to measure accelerations of the device in three orthogonal directions. Sensors to measure environmental conditions and skin conductivity are placed into the cover of the box.

Device used to follow users

The research paper does not specify the mobile device used for the project.

A.4.3 Dataset information

Length

Length of tests is of c.250 entries in total and 2 people. The same 5 scenarios where repeated 40 to 50 times.

Information captured

The data that was sensed was the position of the device, environment's light and sound, pressure and speed.

Sample dataset Table 32 Dataset of the context recognition project

Scenario number	1	1	1	1	1	1	1	1
Repetition number	3	3	3	3	3	3	3	3
Time (s)	0	1	2	3	4	5	6	7
Device:Position:DisplayDown	0.504	0.504	0.895	0.916	0	0	0	0
Device:Position:DisplayUp	0	0	0	0	0	0	0	0
Device:Position:AntennaDown	0	0	0	0	0	0	0	0
Device:Position:AntennaUp	0.662	0.662	0	0	0.808	0.768	0.808	0.841
Device:Position:SidewaysRight	0	0	0	0	0	0.116	0	0
Device:Position:SidewaysLeft	0	0	0	0	0	0	0	0
Device:Stability:Stable	1	1	0	0	0	0	0	0
Device:Stability:Unstable	0	0	1	1	1	1	1	1
Device:Placement:AtHand	0	0	1	1	1	1	1	0
Environment:Light:EU	1	1	1	1	1	1	1	1
Environment:Light:USA	0	0	0	0	0	0	0	0
Environment:Light:Bright	0	0	0	0	0	0	0	0
Environment:Light:Normal	1	1	1	1	1	1	1	1
Environment:Light:Dark	0	0	0	0	0	0	0	0
Environment:Light:Natural	0	0	0	0	0	0	0	0
Environment:Light:TotalDarkness	0	0	0	0	0	0	0	0
Environment:Temperature:Hot	0	0	0	0	0	0	0	0
Environment:Temperature:Warm	0.867	0.867	0.87	0.871	0.872	0.873	0.876	0.878
Environment:Temperature:Cool	0.133	0.133	0.13	0.129	0.128	0.127	0.124	0.122
Environment:Temperature:Cold	0	0	0	0	0	0	0	0

Environment:Humidity:Humid	1	1	1	1	1	1	1	1
Environment:Humidity:Normal	0	0	0	0	0	0	0	0
Environment:Humidity:Dry	0	0	0	0	0	0	0	0
Environment:SoundPressure:Silent	0.731	0.726	0.685	0.732	0.701	0.727	0.626	0.733
Environment:SoundPressure:Modest	0.269	0.274	0.315	0.268	0.299	0.273	0.374	0.267
Environment:SoundPressure:Loud	0	0	0	0	0	0	0	0
UserAction:Movement:Walking	0	0	0	0.65	0.779	0.795	1	1
UserAction:Movement:WalkingFast	0	0	0	0	0	0	0	0
UserAction:Movement:Running	0	0	0	0	0	0	0	0

Table 32 Dataset of the context recognition project

A.5 Project selection and dataset analysis

On our analysis, we apply the framework described earlier and assess each project according to the evaluation criteria (Table 33).

	Reality Mining	Context Dataset	Context recognition
Description	Record of social interaction based on the interpretation of data related to behavior when using the mobile device.	Record of regular daily life of the subjects, routes, work and leisure.	Record of sensor signal data focused on monitoring user activity and environment of real life.
Data	Phone logs for communication includes time, duration, recipient and status. Bluetooth connections, devices and time stamp and location.	Phone logs for communication includes calls and text messages, location, profile and calendar. Set contains many gaps.	Data captured is related to movement, walking, static, up and down. It is captured promptly by repeating an activity 25 times.
Infrastructure	Nokia 6600 cell phones Bluetooth enabled. ContextLog application always running.	Nokia 7650 devices. ContextLog application not always running.	External sensor box linked to a laptop.

Table 33 Project overview

We now analyze the projects to find out which one has the longest traces, more interest for our research (variety of context variables) and data quality on Table 34.

	Reality Mining	Context Dataset	Context recognition
History length	High – 9 months and 94 people.	Low – 25 people and non specified period of time.	Low – 5 scenarios and 25 data entries lasting 3 minutes each.
Interest	High – Communication log is very complete.	High – Communication log is very complete.	Low – Captured data is focused on action only.
Data quality	High – Dataset includes quality check.	Low – There are gaps on the data with no quality check performed to it.	Low – No quality check is done.

Table 34 Project assessment

The selected dataset is the Reality Mining project since it has the most complete dataset and the data it contains is the most aligned to our research objectives. We perform an in -depth analysis of this set, with the goal of understanding its size and the type of data it contains. The Reality Mining project dataset will allow us to perform empirical evaluation of mobile user behavior, which is an increasingly relevant academic research topic (Kivi 2007). Up to now, it had been difficult to studies using real or empiric data (Kivi 2007) used probably due to the lack of real mobility traces (Song, Kotz et al. 2006).

When analyzing the chosen training set, we focus on the nature of its data: size of the traces, variety of symbols they contain, and the time frame of the project. When performing our analysis, we consider the context variables that can be modeled into symbols that are related to user behavior with the mobile device, since the objective of this thesis is to improve mobile communications.

As a result we will select those data traces that have richer information and that will be more interesting to use as input traces for the machine learning algorithms we use for context awareness and prediction.

When ranking the duration of each user in the project, we can see the Table 35.

Trace	Average days
025	300+ days
053	250-300 days
020	250-300 days
093	250-300 days
004	250-300 days
012	250-300 days
023	250-300 days
060	250-300 days
102	250-300 days
036	250-300 days
030	250-300 days
008	250-300 days
041	250-300 days

Table 35 Top 10 longest user traces

We can see a user percentage distribution of the dates of the project in Figure 117.

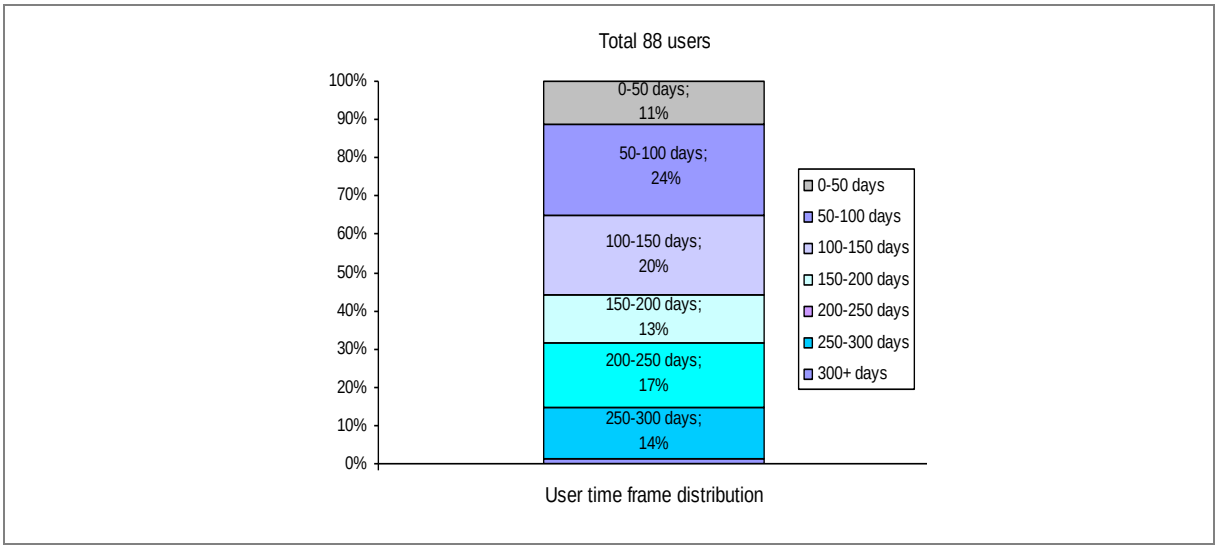

Figure 117 User time frame distribution

We analyze the three groups of data traces context variables (communication, location and applications used) to understand the **size** of the datasets and the amount and type of **activity** it contains.

A.5.1 Communication dataset analysis

Based on our context filtering, a user trace is a voice communication log of calls from a specific user, either outgoing or incoming.

We analyze how long users stay in the project measuring by the total voice calls that occur (either incoming, outgoing or missed) and how many are distinct.

- **size:** number of events that dataset have, it ranges from 142 to 7750 logs, with an average of 2006, which we can see in Figure 118.

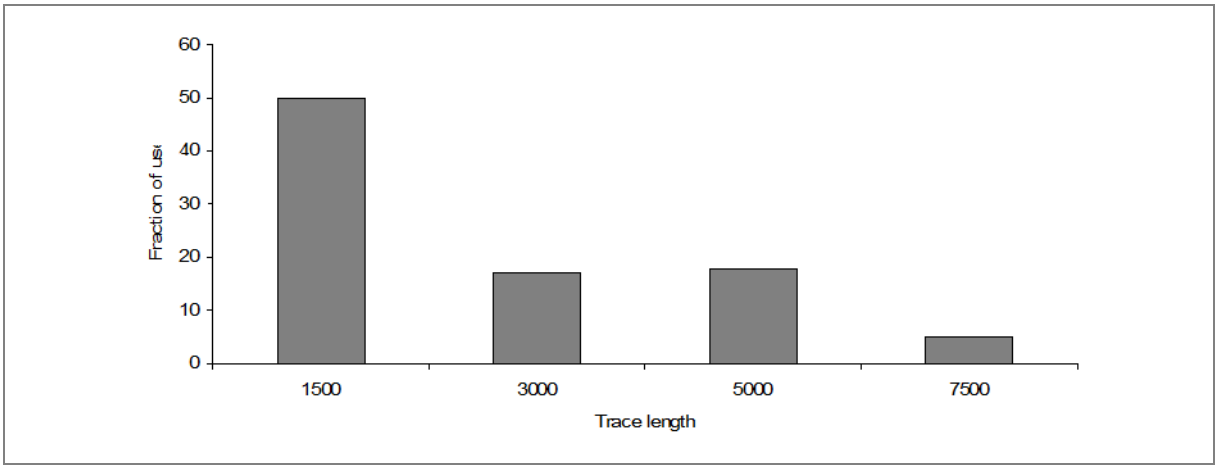

Figure 118 Dataset trace size split per user

- **activity**: how active the students calls have been based on the number of distinct calls they have done and received. The average of distinct calls is 90, with a range from 2 to 194, which we can see in Figure 119.

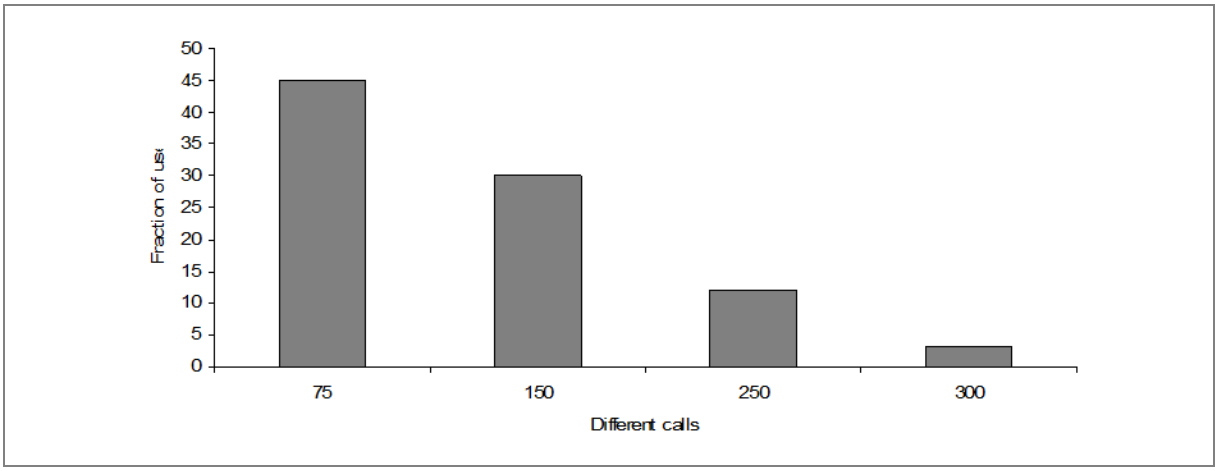

Figure 119 Number of distinct voice calls user make

A.5.1 Location dataset analysis

Based on our context filtering, a user trace is a location log from a specific user, filtering those that are void or repeated.

We analyze how long users stay in the project measuring it number of days, the total cells on which the mobile device has been connected to.

- **size**: number of events that dataset have, it ranges from 229 to 59132 logs, with an average of 36854, which we can see in Figure 120.

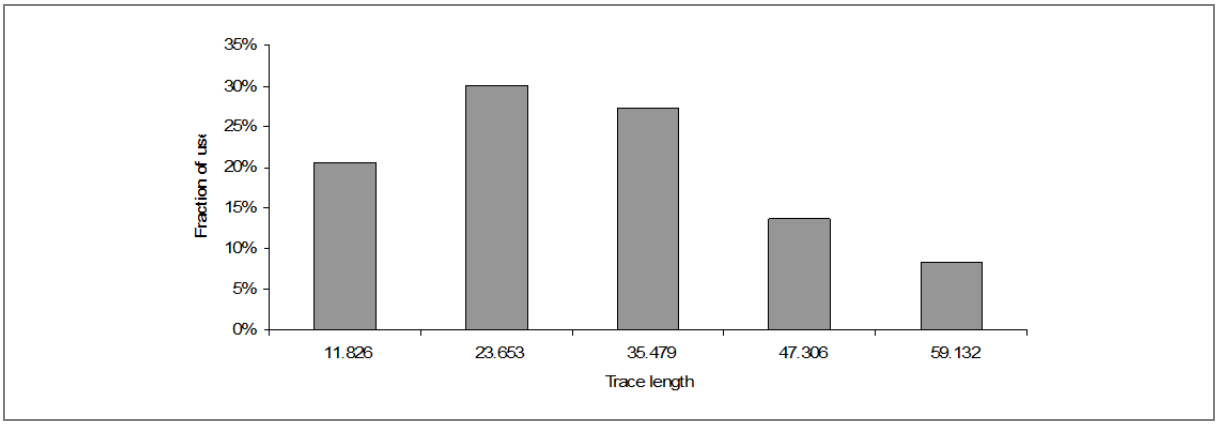

Figure 120 Dataset traces split per user

- **activity**: how active the students are in the campus and how much do they move from one cell area to another. The average of distinct cells is 1655, with a range from 17 to 3137, which we can see in Figure 121.

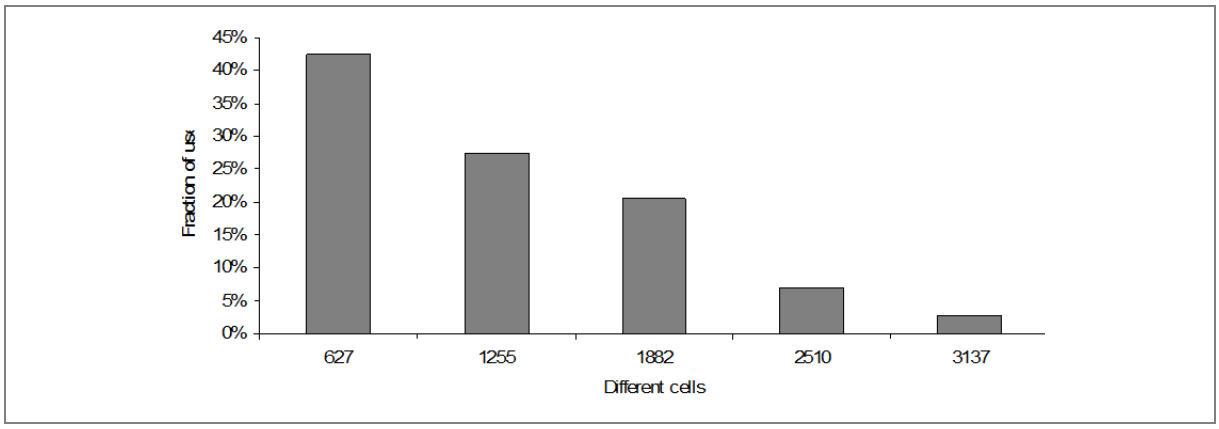

Figure 121 Number of distinct cells the phones connect to

A.5.2 Applications dataset analysis

Based on our context filtering, a user trace is an application log from a specific user, filtering those that are void.

We analyze the total applications have been in use on the mobile device and the distinct ones.

- **size**: number of events that datasets have, it ranges from 70 to 24783 142 to 7750 logs, with an average of 7922, which we can see in Figure 122.

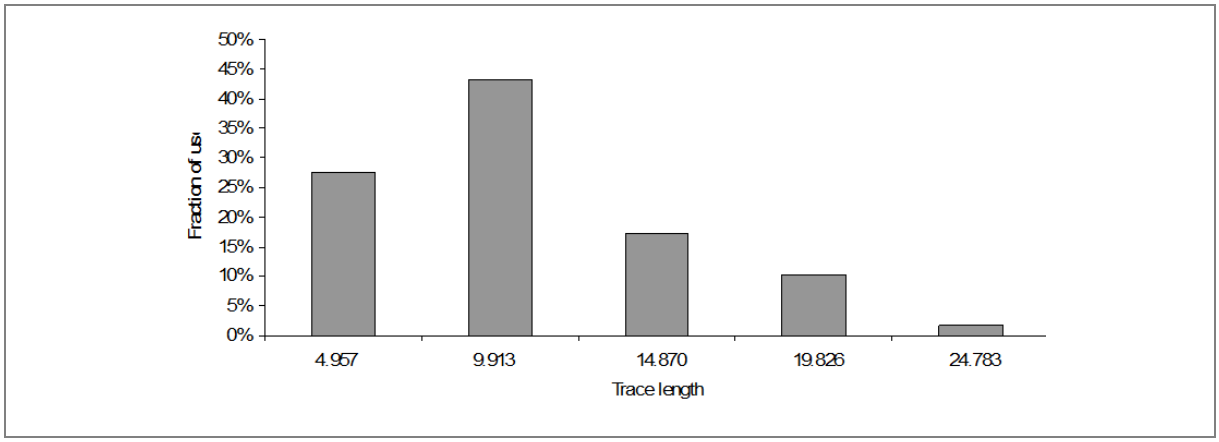

Figure 122 Dataset traces size split per user

- **activity**: amount of distinct phone applications students use in the mobile device. The average of distinct applications is 19 with a range from 8 to 23, which we can see in Figure 123.

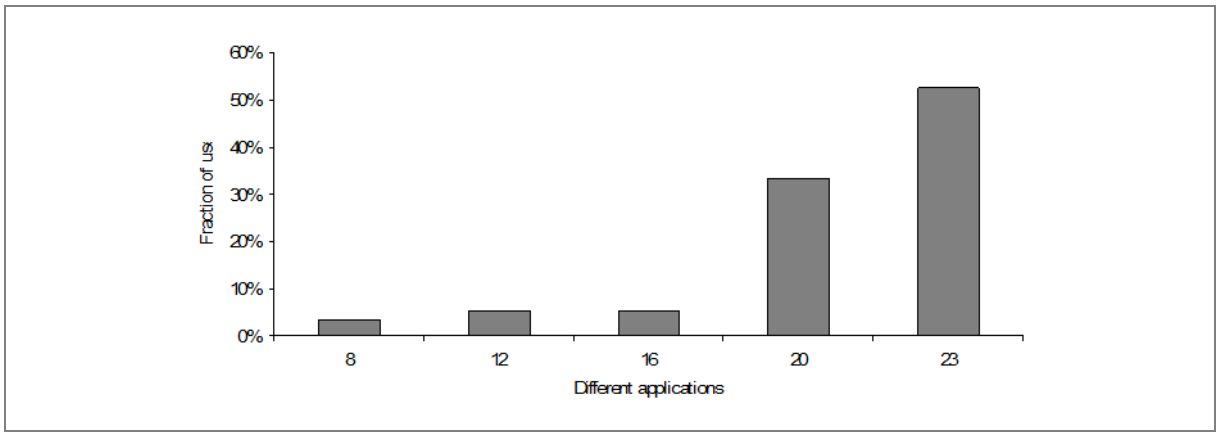

Figure 123 Number of distinct cells the phones connect to

Based on our analysis we will select a subset of the data traces that has enough information to be used as a test set and that can be modeled in a stochastic manner to serve as input for the learning algorithm. We blend the top 10 data traces of each context variables with the largest time frame in the project (Table 36).

- 10 largest communication data traces: 20, 04, 52, 67, 40, 36, 53, 08, 81, 23
- 10 largest location data traces: 04, 74, 12, 65, 99, 86, 08, 22, 81, 70
- 10 largest application data traces: 20, 93, 36, 23, 22, 53, 52, 08, 21,12
- 10 longest data traces: 25, 53, 20, 93, 04, 12, 23, 60, 102, 36, 30, 08, 41.

Communication		
Trace	Events	Symbols
Trace020	7750	294
Trace004	6135	100
Trace052	5689	280
Trace067	5572	160
Trace040	5544	155
Trace036	4972	184
Trace053	4852	214
Trace008	4753	158
Trace081	4572	165
Trace023	4165	161

Locations		
Trace	Events	Symbols
Trace004	59132	1744
Trace074	58624	1774
Trace012	56458	3137
Trace065	56454	697
Trace099	54962	2172
Trace086	49597	2031
Trace008	45775	2794
Trace022	45546	2430
Trace081	45094	1751
Trace070	45035	2237

Applications		
Trace	Events	Symbols
Trace020	24783	20
Trace093	16583	22
Trace036	16433	21
Trace023	16220	21
Trace022	15003	19
Trace053	14955	20
Trace052	14275	20
Trace008	13035	18
Trace021	13032	22
Trace012	12628	23

Project	
Trace	Days
Trace025	311
Trace053	296
Trace020	293
Trace093	277
Trace004	277
Trace012	277
Trace023	277
Trace060	277
Trace102	275
Trace036	272

Table 36 Top 10 most active and largest data traces

The resulting traces to be used as input for the learning algorithm are: Trace04, Trace08, Trace12, Trace20, Trace22, Trace23, Trace36, Trace53, Trace81, and Trace93 since they have the largest and most relevant amount of information.

Appendix B: Program description

In our context-aware mobile architecture we develop a series of programs that capture context variables, process then and predict context states. On this section we describe some of the different programs we have implemented throughout the thesis to extract context data, model context into a unified context vector and to predict future context signals. Most of the code is programmed in python except the data base query to extract data from the project's data traces and save it into a text file, which is a Matlab script.

B.1. Dataset variables extraction Matlab scripts

We extract the data from the Matlab data base through scripts that select and store data into a text files.

1. Open Matlab and switch current directory to the one that stores the data base and scripts.
2. Import data base into the workspace window, "File-->Import Data", choosing the data base and select all variables from the import wizard (network and s).
3. Variables can be analyzed by double clicking on them.
4. To run the script, type on Matlab's command window "nameScript(s)"

We obtain all data contained in the data base by each variable group by creating a script to extract data from the Matlab data base, we can see an example of the code of one of the scripts in Figure 124.

```
function commlog2file(s)
for i=1:length(s(1,:))
   name = ['.\\trace' sprintf('%03d',i) '.txt'];
   fid = fopen(name, 'wt');
   if fid~=-1
      if length(s(1,i).comm)>0
         for j=1:length(s(1,i).comm)
            fprintf(fid,'%6.4f\t%d\t%d\t%s\t%s\t%d\t%d\n',s(1,i).comm(1,j).date,s(1,i).comm(1,j).event,
s(1,i).comm(1,j).contact,s(1,i).comm(1,j).description,s(1,i).comm(1,j).direction,s(1,i).comm(1,j).duration,
s(1,i).comm(1,j).hashNum);
         end
      end
      fclose(fid);
     else
   end
end
```

Figure 124 Matlab script to extract context data

Once the data is extracted, it is kept on text files, one per user and per type of context data. We see a sample of the app context data file log in Figure 125.

03-Aug-2004 11:30:04	context_log
03-Aug-2004 11:30:10	Menu
03-Aug-2004 11:50:01	context_log
03-Aug-2004 11:50:12	Phone
03-Aug-2004 11:53:09	context_log
03-Aug-2004 12:26:40	Phone
03-Aug-2004 12:37:59	Phone
03-Aug-2004 13:15:59	context_log
03-Aug-2004 13:16:49	Phone
03-Aug-2004 13:27:00	Phone
03-Aug-2004 13:27:04	context_log
03-Aug-2004 13:54:43	context_log
03-Aug-2004 13:56:13	Menu
03-Aug-2004 13:56:16	Phone
03-Aug-2004 13:56:18	context_log
03-Aug-2004 13:57:01	Phone
03-Aug-2004 13:57:02	Menu
03-Aug-2004 13:57:11	Menu
03-Aug-2004 13:57:27	Phone
03-Aug-2004 13:57:29	context_log
03-Aug-2004 15:12:08	context_log
03-Aug-2004 15:12:18	Menu

Figure 125 Sample of context log for apps

We then need to clean the data, eliminating null values and formatting all the date variables as dates for which we create a program to do this and look for rows with all 0 values on it and changes months that are written in text format to a numeric one from 1 to 12.

B.2. Context modeling program code

This program reads the training sets, analyzes and understands their data to transform it into a format that the learning algorithm will take as input.

This is done in two phases,

- Phase 1: extract data from the data text file and transform it into a context variable matrix. We define a matrix structure that stores all the possible context data, storing a blank when there are no values or that variable.
- Phase 2: filter data from the data context variable matrix into different input files for the learning algorithm, based on the type of information we will like to analyze.

The program reads files and extracts the available content data storing it into a context vectors, one per each group of selected data from the context value matrix.

It is based in the following functions

- **Read data trace file and transform it:** Functions that open the project's text file and scans through each line parsing it through its blanks (Figure 126 and Figure 127).

  Input is traceXXX.txt which is a text file, where XXX= 001 to 106.

  Output is a list of parsed tags TagList.

Transform data: DataTransformation.py

```
import sys/ re /  getopt
import ParseFile #read file and parse it
import CreateMatrix #create a matrix with all the context values
import MatrixFile #create a file with the matrix values
def main():
###initiate variables
DataSet_Lines =  ContextMatrix = []
DataSet_NumLines = 0
DataTrace_Number = ""
DataSet_Lines, DataTrace_Number =  ParseFile.ParseFile() #Open list of lines to parse tags
DataSet_NumLines = len (DataSet_Lines)
ContextMatrix=CreateMatrix.CreateMatrix(DataSet_Lines,DataSet_NumLines)
MatrixFile.MatrixFile(ContextMatrix, DataTrace_Number)
```

Figure 126 Pseudo-code for data transformation

Read file: ParseFile.py

```
import sys/ re
def ParseFile ():
TagList = [] #list where I will keep the tags
 line = "" #string where I keep each line I read
fileName = "" #name of the file
traceNumber = "" #number of data trace
 fileName = 'trace001.txt'
traceNumber = fileName[5:8]
 file = open(fileName, 'r')
 for line in file:   ## iterates over lines in file
line.split()
TagList.append(line)
file.close()
return TagList, traceNumber
```

Figure 127 Pseudo-code for data parsing

- **Create context matrix:** Reads the parsed tags, and stores its value into a matrix of context vectors, as we can see in Figure 128.

  Input is the number of lines to read, NumLines, and the tags values of the dataset, Dataset Tags.

  Output is a matrix containing the context vectors MatrixVector and its length, Matrix Rows.

Create matrix: CreateMatrix.py

```
import sys/ re/ string
DataSet_Values = []
NumLines = 0
def CreateMatrix (DataSet_Values, NumLines):
#TagList = [] #list where I will keep the tags
i = 0 #counter
line = "" #string that stores each line read
TempVector =     MatrixVectors = []
#context data vectors definition
VectorContext = VectorDate = VectorTime = VectorEventID = VectorContact =  VectorAction = VectorDirection =
VectorDuration =  VectorReceiver  = []
while (i < NumLines):
 line = DataSet_Values[i].split()
 VectorContext.append('comm')
      VectorDate.append(line[0] toline[3])
      if (line[4] == 'Voice'):
         VectorAction.append('v')
      elif (line[4] == 'Short'):
         VectorAction.append('s')
      else: #its a packet data
         VectorAction.append('p')
      if (line[6] == 'Incoming'):
         VectorDirection.append('i')
      elif (line[6] == 'Outgoing'):
         VectorDirection.append('o')
      else:
         VectorDirection.append('m')
      VectorDuration.append(line[7])
      if (VectorDirection[i] == 'm'):
         VectorReceiver.append('')
      else:
         pass
i = i +1
   MatrixVectors = [VectorContext, VectorDate, VectorTime, VectorEventID, VectorContact, VectorAction,
VectorDirection, VectorDuration, VectorReceiver]
return MatrixVectors
```

Figure 128 Pseudo-code program that creates context vector

- **Write matrix into a file: Write the matrix's vector's context data into a text file.**

  Input Context matrix and trace file name.

  Output file MatrixXXX.txt, where XXX is the trace number**.**

B.3. Predicting context signals

Three steps: Model into symbols, input on the prediction algorithm and output a result file.

1. Model context logs into into symbols

We create a program that converts the data on the context logs into symbols that can be read by the learning algorithm, the symbols are created dynamically, storing all the context

data values into a table and naming them with an alphanumeric value. We can see a sample of how these files look like on Figure 129.

Application symbols
M1
F1
P1
P1
P1
M1
M1
P1
P1
P1
P1
M1
P1
P1
P1
P1
P1
C1
P1
P1
M1
A1
M1
H1

Figure 129 Sample of symbol contexts log files

2. Markov Model prediction algorithm

To illustrate how the algorithm works, we analyze the example input string abadcadcabcbdcab.

The algorithm first builds a tree that will support the algorithm the depth of the tree is dependent on its order, which is fixed manually. The order of the predictor is the order of the context and it represents the size of the past that has been seen, i.e. the number of precedent events (Katsaros and Manolopoulos 2005). We can see how the tree looks in Figure 130.

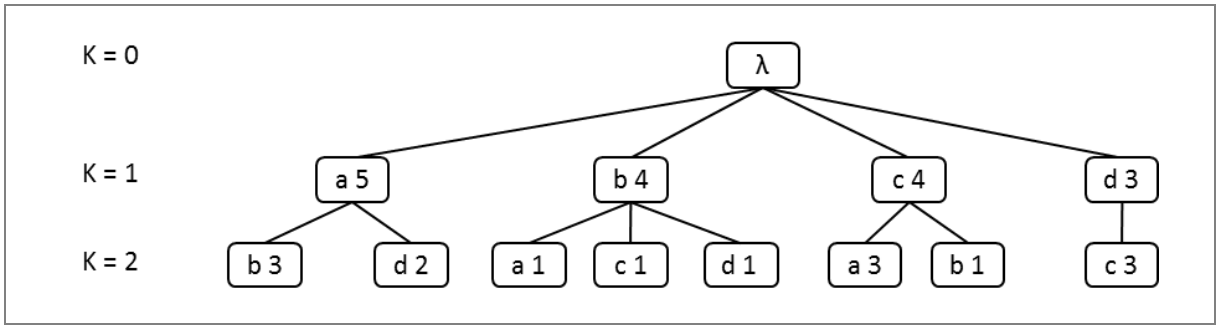

Figure 130 Trie representation of the string abadcadcabcbdcab

We then calculate the frequencies of each context up to the maximum order which we fixed in 2 (Table 37).

k = 0	
ctx	N(λ,string)
a	5
b	4
c	4
d	3

k = 1				
ctx	N(a,string)	N(b,string)	N(c,string)	N(d,string)
a	0	1	3	0
b	3	0	1	0
c	0	1	0	3
d	2	1	0	0

k = 2								
ctx	N(ab,string)	N(ad,string)	N(ba,string)	N(bc,string)	N(bd,string)	N(ca,string)	N(cb,string)	N(dc,string)
a	1	0	0	0	0	0	0	3
b	0	0	0	1	0	2	0	0
c	1	2	0	0	1	0	0	0
d	0	0	1	0	0	1	1	0

Table 37 ContextCount. Frequency count of all the contexts over the string abadcadcabcbdcab

Based on the table it creates a state matrix on which each entry represents the probability of changing from one symbol to another given a specific context (Figure 131.)

$$M= \begin{pmatrix} P(a,a) & P(b,a) & P(c,a) & P(d,a) \\ P(a,b) & P(b,b) & P(c,b) & P(d,b) \\ P(a,c) & P(b,c) & P(c,c) & P(d,c) \\ P(a,d) & P(b,d) & P(c,d) & P(d,d) \end{pmatrix}$$

Figure 131 Transition states matrix

The transition inside the matrix is defined Equation 9.

$$P(i,j) = P(X_{n+1} = S_i \mid ctx = S_{i-1})$$

Equation 9 Transition matrix state calculation

Where,

P = probability

i, j = states

ctx = context

s_i = next symbol

s = symbol $_{i-1}$

In order to implement it, we create a program called “algorithmML.py” , we can see its pseudo-code in Figure 132.

algorithmML.py
Create matrix table to store the symbols and appearance count
Define a string variable to store the symbol log read
Define the variable to store the algorithm’s order
Create the prediction vector and prediction node
Define the probability formula for the algorithm using Equation 4
Open up the symbol files and store it into the symbol string
Create the matrix table first row
While the whole symbol string is read:
Parse each symbol, read it and store it in the matrix table
Update the number of times the symbol appears
Calculate the probability of the upcoming symbol
Store the predicted upcoming symbol in the matrix
Read the upcoming symbol from the symbol string and check if it matches the predicted one
Store the prediction value, 1 for hit, 0 for error

Figure 132 Pseudo-code program that creates context vector

3. Output file

We can see an example of the output of the Markov Model prediction algorithm in Figure 133.

Example: User 004

Input trace		Trace004		
Algorithm:		MM		
Order:	1			
Entries:	10728			
Tree depth:		2		
Number nodes:		20		
Different symbols:		19		
Run time:		235.55		
Num guesses:		10727		
Average guess:		0.543		
Show results:		100		
Count	#read	Prob	Guess	Nodes
100	101	0.364	0.53	7.0
200	201	0.374	0.51	10.0
300	301	0.377	0.44	12.0
400	401	0.545	0.56	13.0
500	501	0.368	0.47	14.0
600	601	0.444	0.65	14.0
700	701	0.368	0.5	14.0
800	801	0.432	0.6	14.0
900	901	0.436	0.52	14.0
1000	1001	0.364	0.46	14.0
1100	1101	0.37	0.6	14.0
1200	1201	0.375	0.67	15.0
1300	1301	0.444	0.54	15.0
1400	1401	0.367	0.43	17.0
1500	1501	0.369	0.55	17.0
1600	1601	0.436	0.54	17.0
1700	1701	0.427	0.47	17.0
1800	1801	0.729	0.49	17.0

Figure 133 Sample of a Markov Model prediction output file

4. slicing the input signals into time frames prior to predicting the results

We include a module that filters the context log files according to a specific time-slice that we change dynamically with the prediction output. Originally, the slots are defined to be: M (morning) = 8-12, A (afternoon) = 12-18, E (evening) = 18-23, and they are changed until they reach the defined prediction threshold.

This functionality is done through two functions: one “FilterFileTimeSlots.py” is in charge of opening the context file and storing it into a new file with its symbols tied to their time frame. The other, “FilterSlotFunction.py” takes this time frame file as input and slices it according to the specified time slot defined. We can see the function pseudo-code in Figure 134.

FilterFileTimeSlots.py	FilterSlotFunction.py
Obtain input file from the program Create output file name as an extension of the input file file = open(fileInputName, 'r') #open file iterate over the file and store values on a list store time, date and context value on a new tag list create an output file with the symbols + time values	Obtain input file from the program Create output file name as an extension of the input file file = open(fileInputName, 'r') #open file iterate over the file and store values on a list store time, date and context value on a new tag list create an output file with the symbols + time values

Figure 134 Pseudo-code for slicing context logs into time frames

Glossary index

3

3G Third generation mobile telecommunications

A

Ads Advertisement

ALZ Active-LeZi data compression algorithm

API Application Programming Interface

App Application

B

Bayes Net Bayesian network probabilistic classifier

B-MAD Bluetooth Mobile Advertising

C

CPU central processing unit

E

DT Decision Table classifier based on logic rules

D

email Electronic mail

F

F1 Score Measure of a test's accuracy

G

GHz Giga Hertz

GNU General Public License

GPRS General Packet Radio Service

GPS Global Positioning System

GSM Global System for Mobile Communications

H

HTML5 HyperText Markup Language

I

IBL Instance Based Learning algorithm

ID Identification

IR Infra-Red

ISP Internet Service Provider

L

LAN Local Area Network

Lazy IB1 Lazy Instant Based classifier

LBS Location Based System

LED Light-Emitting Diode

LTE Long Term Evolution

LZU Universal lossless data compression algorithm

M

MAN Metropolitan Area Network

MB Mega Byte

MHz Mega Hertz

MIT Massachusetts Institute of Technology

MM Markov Model

N

Naïve Bayes simple probabilistic classifier based Bayes' theorem

NAND Non-volatile computer memory

NFC Near Field Communication

O

OS Operating System

PC Personal Computer

PDA Personal Digital Assistant

PIM Personal Information Manager

RAM Random-access memory

RFID Radio frequency identification

RIM Research In Motion

SDK Software Development Kit

SERP Search engine results page,

SMS Short Message Service

SQL Structured Query Language

S

SVM Support Vector Machine algorithm

T

TV Television

U

UMTS Universal Mobile Telecommunications System

W

WAN Wide Area Network

WAP Wireless Application Protocol

WEKA Waikato Environment for Knowledge Analysis

Wi-Fi Wireless Fidelity

WiMAX Worldwide Interoperability for Microwave Access

WLAN Wireless Local Area Network

X

XHTML Extensible HyperText Markup Language

XML Extensible Markup Language

Bibliography

Aalto, L., N. Göthlin, et al. (2004). Bluetooth and WAP push based location-aware mobile advertising system. International Conference On Mobile Systems, Applications And Services. Boston, MA, USA, ACM. Proceedings of the 2nd international conference on Mobile systems, applications, and services: Pages: 49 - 58

Abraham, A. (2005). "130: Rule-based Expert Systems."

Acer. (2012). "Acer homepage." Retrieved May 2012, from http://www.aceronline.es/shop/acer-aspire-one-c-54_100.html.

Adelstein, F., S. K. S. Gupta, et al. (2005). Fundamentals of mobile and pervasive computing.

Aha, D. w., D. Kibler, et al. (1991). "Instance-Based Learning Algorithms." Machine Learning 6(37-66).

Albrecht Schmidt, Michael Beigl, et al. (1999). "There is more to context than location " Computers & Graphics Elsevier 23(6): 893-901.

Andreeva, P., M. Dimitrova, et al. (2004). "Data mining learning models and algorithms for medical applications."

Anumba, C. and Z. Aziz (2006). "Case Studies of Intelligent Context-Aware Services Delivery in AEC/FM." Springer 4200/2006(LNCS: Intelligent Computing in Engineering and Architecture): 23-31.

Arthur, S. (2007). Some Studies in Machine Learning Using the Game of Checkers. IBM Journal 1959-03-03: 210–229.

Ashkan, A., C. L. A. Clarke, et al. (2009). Classifying and Characterizing Query Intent. ECIR: pp. 578-586

Bardram, J. E. (2004). Applications of ContextAware Computing in Hospital Work – Examples and Design Principles. ACM Symposium on Applied Computing.

Barkhuus, L. and A. Dey (2003). Is context-aware computing taking control away from the user? Three levels of interactivity examined. Proceedings of Ubicomp Seattle, WA.

Begleiter, R., R. El-Yaniv, et al. (2004). "On prediction using variable order Markov models." Journal of Artificial Intelligence Research 22(1): 385-421.

Bernardos, A. M., P. Tarrío, et al. (2008). A data fusion framework for context-aware mobile services. . Proceedings of the IEEE International Conf. in Multisensor Fusion and Integration for Intelligent Systems, IEEE Computer Society: 606-613,.

Bifet, A., C. Castillo, et al. (2005). "An Analysis of Factors Used in Search Engine Ranking."

Bilton, N. (2011). "The Sensors Are Coming!" Technology, from http://bits.blogs.nytimes.com/2011/05/19/the-sensors-are-coming/.

Bloem, A., K. Thomas, et al. (2009). The search and information access report. C. watch.

Bouckaert, R. R., E. Frank, et al. (2010). WEKA Manual for Version 3-6-4.

Bouquet, P., L. Serafini, et al. (2008). Perspectives on contexts, CSL Publications.

Brin, S. and L. Page (1998). "The Anatomy of a Large-Scale Hypertextual Web Search Engine." Computer Networks and ISDN Systems 30(1-7): 107-117.

Broder, A. (2002). A taxonomy of web search A. S. Forum. 36.

Broder, A., M. Fontoura, et al. (2007). "A Semantic Approach to Contextual Advertising."

Brown, P. J., J. D. Bovey, et al. (1997). "Context-aware applications: from the laboratory to the marketplace." IEEE Personal Communications 4(5): 58-64.

Buchholz, T., A. Kupper, et al. (2003). Quality of context: What it is and why we need it. . Proceedings of the Workshop of HP OpenView University Association, Geneva, HP.

CNET. (2012). "Dell Latitude E4200 " Retrieved July 4th, 2012, from http://asia.cnet.com/product/dell-latitude-e4200-core-2-duo-ulv-su9400-processor-1-4ghz-2gb-ram_specs-43888807.htm.

Corp., i. (2010, 5 May 2010). "Shipments of Cell Phone Motion Sensors to Rise Fivefold by 2014." Retrieved 22 May 2010, 2010, from http://www.cellular-news.com/story/43168.php.

Coutaz, J. and J. L. Crowley (2005). "Context is key." Communication of ACM 48(3): 49 - 53.

Cristianini, N. and J. Shawe-Taylor (2000). An introduction to support vector machines and other kernel-based learning methods. Cambridge, Cambridge University Press.

Cufoglu, A., M. Lohi, et al. (2008). "Classification Accuracy Performance of Naïve Bayesian (NB), Bayesian Networks (BN), Lazy Learning of Bayesian Rules (LBR) and Instance-Based Learner (IB1) – Comparative Study." IEEE

Chan, A. T. S. and S.-N. Chuang (2003). "MobiPADS: A Reflective Middleware for Context-Aware Mobile Computing." IEEE Transactions on Software Engineering, 29(12).

Chaoming Song, Zehui Qu, et al. (2010). "Limits of Predictability in Human Mobility." Science 327 (5968): 1018-1021

Chen, G. and D. Kotz (2000). A Survey of Context-Aware Mobile Computing Research. Technical Report, Darmouth College.

Chen, G. and D. Kotz (2002). Solar: A pervasive-computing infrastructure for context-aware mobile applications. Dartmouth Computer Science Technical Report TR2002-421, Dartmouth College. .

Cheng, J. and R. Greiner (1999). Comparing Bayesian Network Classifiers. UAI'99 Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence 101-108

Church, K. and B. Smith (2007). "Mobile Information Access: A Study of Emerging Search Behavior on the Mobile Internet." ACM Transactions on the Web 1(4).

Church, K. and B. Smith (2008). "Who, what, where and when: A new apporach to mobile search " ACM

Dargie, W. (2009). Context and self-management. Context-aware computing and self-management systems. W. Dargie. Boca Raton, FL, Chapman & hall book 5-11.

Das, S. K., D. J. Cook, et al. (2002). "The role of prediction algorithms in the MavHome smart home architecture." IEEE Wireless Communications December 1536-1284.

Dey, A. K. (2001). "Understanding and Using Context." Personal and Ubiquitous Computing, Springer 5 (1): 4 -7

Dey, A. K. and G. D. Abowd (1999). Towards a Better Understanding of Context and Context-Awareness. 1st International Symposium on Handheld and Ubiquitous Computing (HUC ÃƒÂ99): 304-307.

Domingos, P. (2012). "A Few Useful Things to Know about Machine Learning." Communications of the ACM 55 (10): 78-87

Eagle, N., A. Pentland, et al. (2009). "Inferring friendship network structure by using mobile phone data." PNAS. vol 106, no. 36.

Feijoo, C., C. Pascu, et al. (2009). "The Next Paradigm Shift in the Mobile Ecosystem: Mobile Social Computing and the Increasing Relevance of Users." COMMUNICATIONS & STRATEGIES 75(3rd quarter): 57-77.

Forstadius, J., O. Lassila, et al. (2005). RDF-based Model for Context-aware Reasoning in Rich Service Environment. Proc. of the 3rd Int. Conf. on Pervasive Computing
and Communications Workshops. IEEE Computer Society.: 15-19.

Franklin, C. (2004). "How Internet Search Engines Work ", from http://computer.howstuffworks.com/internet/basics/search-engine1.htm.

Friedman, N. and M. GoldSzmidt (1996). Building classifiers using bayesian networks. AAAI'96 Proceedings of the thirteenth national conference on Artificial. 2: 1277-1284.

Gartner. (2012). "Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth." from http://www.gartner.com/it/page.jsp?id=1924314.

Gellersen, H. W., A. Schmidt, et al. (2002). "Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts." Mobile Networks and Applications 7(5): 341–351.

Gómez-Barroso, J. L., R. Compañó, et al. (2010). Prospects of Mobile Search, IPTS.

Gopalratnam, K. and D. J. Cook (2003). Active LeZi: An Incremental Parsing Algorithm for Sequential Prediction. Proceedings of the Florida Artificial Intelligence Research Symposium, Florida.

Gopalratnam, K. and D. J. Cook (2007). "Online sequential prediction via incremental parsing: the active LeZi Algorithm." IEEE Intelligent Systems 22(1): 52-58

Grauballe, A., G. P. Perrucci, et al. (2008). Introducing Contextual Information to Mobile Phones by External and Embedded Sensors. International workshop on mobile device and urban sensing MODUS08. St. Louis,MO

Gruber, T. R. (1992). "Toward Principles for the Design of Ontologies Used for Knowledge Sharing." International Journal Human-Computer Studies 43 907-928.

Gu, T., X. H. Wang, et al. (2004). An Ontology-based Context Model in Intelligent Environments. PROCEEDINGS OF COMMUNICATION NETWORKS AND DISTRIBUTED SYSTEMS MODELING AND SIMULATION CONFERENCE.

Heimonen, T. and M. Käki (2007). Mobile findex - supporting mobile web search. International conference on Human Computer Interaction with Mobile Devices and Services.

Henricksen, K. and J. Indulska (2004). A Software Engineering Framework for Context-Aware Pervasive Computing. Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (PERCOM'04)

Henricksen, K., J. Indulska, et al. (2002). Modeling Context Information in Pervasive Computing Systems. Proceedings of 1st Int. Conf. Pervasive Computing, LNCS 2414,
, Springer.

Hinzen, W. (2008). Context and logical form. Perspectives on contexts,. L. S. Paolo Bouquet, Richmond H.Thomason, CSL Publications.: 189-215.

Holmes, G., A. Donkin, et al. (1994). WEKA: A Machine Learning Workbench. C. S. W. Papers, Hamilton, New Zealand: University of Waikato, Department of Computer Science.

Hsu, C.-W., C.-C. Chang, et al. (2003). "A practical guide to support vector classification."

Inc, M. (2011). "AroundMe descirption." iPhone Retrieved May 20 2012, from http://www.appstorehq.com/aroundme-iphone-5119/app.

Juniper (2009). Mobile augmented reality. A whole new world.

Kamvar, M., M. Kellar, et al. (2009). Computers and iPhones and Mobile Phones, oh my! A logs-based comparison of search users on different devices. www 2009. Madrid, Spain. Track: User Interfaces and Mobile Web / Session: Mobile Web.

Katsaros, D. and Y. Manolopoulos (2005). "A Suffix Tree Based Prediction Scheme for Pervasive Computing Environments." Lecture Notes in Computer Science, Springer-Verlag 3746 267–277.

Kim, E., S. Helal, et al. (2010). "Human Activity Recognition and Pattern Discovery." IEEE Pervasive Computing vol. 9, no. 1: 48-53.

Kivi, A. (2007). Measuring Mobile User Behavior and Service Usage: Methods, Measurement Points, and Future Outlook. Proceedings of the 6th Global Mobility Roundtable. Los Angeles, California, U.S.

Kohavi, R. (1995). The power of decision tables. European Conference on Machine Learning.

Kolmonen, L. (2008). MOBILE SEARCH ENGINE SURVEY. Research Seminar on Telecommunications Business.

Korpipää, P. and J. Mäntyjärvi (2003). "An Ontology for Mobile Device Sensor-Based Context Awareness." Springer-Verlag: 451-458.

Kotsiantis, S. B. (2007). "Supervised Machine Learning: A Review of Classification Techniques." Informatica 31: 249-268.

Krum, C. (2011). "Smartphone Search Engine Crawlers & Mobile URLs." Retrieved May 5th, 2012, from http://searchenginewatch.com/article/2170032/Smartphone-Search-Engine-Crawlers-Mobile-URLs.

Kurkovsky, S., V. Zanev, et al. (2005). "SMMART, a Context-Aware Mobile Marketing Application: Experiences and Lessons. ." Embedded and Ubiquitous Computing, LNCS, Springer. 3823/2005,: 141-150.

Lapkin, A. (2009) "Gartner Says Context-Aware Computing Will Provide Significant Competitive Advantage." Press room Volume, DOI:

Lassila, O. and D. Khushraj (2005). Contextualizing Applications via Semantic Middleware. Proceedings of the Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'05) 2005 IEEE.

Leichtenstern, K., A. D. Luca, et al. (2005). Analysis of Built-in Mobile Phone Sensos for Supporting Interactions with the Real World. Pervasive Mobile interaction workshop 2.

Mäntyjärvi, J., J. Himberg, et al. (2004). Sensor signal data set for exploring context recognition of mobile devices. Benchmarks and a database for context recognition' in conjunction with the 2nd international conference on pervasive computing (PERVASIVE 2004),. Linz/Vienna, Austria.

Mark Hall, E. F., Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009). The WEKA Data Mining Software: An Update; , SIGKDD Explorations. 11.

Massimo Benerecetti, P. B., Chiara Ghindi (2008). On the dimensions of context dependence. Perspectives on contexts,. L. S. Paolo Bouquet, Richmond H.Thomason, CSL Publications.: 189-215.

Mayrhofer, R., H. Radi, et al. (2003). Recognizing and Predicting Context by Learning from User Behavior. the International Conference on advances in mobile multimedia. 171: 25-35.

Mei, T. and X.-S. Hua (2010). Contextual Internet Multimedia Advertising. Proceedings of the IEEE.

Merriam-Webster. (2008). "Definition of 'context'." Retrieved September 6th, 2008, from http://www.merriam-webster.com.

Merriam-Webster. (2010). "Definition of 'learning'." Retrieved May 6th, 2010, from http://www.merriam-webster.com.

Meyer, S. and A. Rakotonirainry (2003). A Survey of Research on Context-Aware Homes. Proceedings of the Australasian information security workshop conference on ACSW frontiers Adelaide, Australia.

Micarell, A., F. Gasparetti, et al. (2007). "Personalized Search on theWorld Wide Web."

Mika Raento, Antti Oulasvirta, et al. (2005). "ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications." IEEE CS and IEEE ComSoc 2005.

Mitchell, T. M. (2006). The Discipline of Machine Learning. Pittsburgh, PA, Carnegie Mellon University.

Nicholas D. Lane, Emiliano Miluzzo, et al. (2010). "A Survey of Mobile Phone Sensing. Communications Magazine." Communications Magazine, IEEE 48 (9): 140 – 150.

Nielsen (2009) "Is Your Mobile Marketing Strategy as Smart as Your Phone?" Net reporter Volume, DOI:

NokiaDeveloper. (2003). "Nokia device specifications." from http://www.developer.nokia.com/Devices/Device_specifications/6600/.

Notess, G. R. (June 8, 2009). "Microsoft's New Bing—The 'Decision Engine' " Retrieved March 1st, 2012, from http://newsbreaks.infotoday.com/NewsBreaks/Microsofts-New-BingThe-Decision-Engine-54514.asp.

Patrick Fahy and S. Clarke (2004). CASS – a middleware for mobile context-aware applications Workshop on Context-Awareness,MobiSys 2004.

PCWorld. (2008). "Acer Aspire One Specs." Reviews» Computers» Laptops Retrieved 04-07-2012, 2012, from http://www.pcworld.com/article/147366/acer_aspire_one_mininotebook.html.

Penco, C. (2008). Context and contract Perspectives on contexts,. L. S. Paolo Bouquet, Richmond H.Thomason, CSL Publications.: 189-215.

Perez, S. (2012). "Local Search App AroundMe Trumps Yelp's Mobile Apps With 6M Monthly Users." Hot topics Retrieved April 10, 2012, from http://techcrunch.com/2012/04/05/local-search-app-aroundme-trumps-yelps-mobile-apps-with-6m-monthly-users/?grcc=7d34439652700cb2345425bd59328f8eZ8ZwdgtZ0Z8Z83Z12Z3.

Powers, D. (2011). "Evaluation: from precision, recall and f-measure to roc, informedness, markedness & correlation." Journal of Machine Learning Technologies 2(1): 37-63.

Pulskamp, F. (2011). Worldwide mobile phone semiconductor 2011-2015 forecast. IDC Market Analysis IDC. Framingham, MA IDC. May 2011.

Python.org. 2011, from http://python.org/.

Quinlan, J. R. (1986). "Induction of decision trees." Machine Learning 1: 81-106.

Raento, M. (2004). Mobile Communication and Context Dataset, Helsinki Institute for Information Technology.

Ramana, B. V., M. S. P. Babu, et al. (2011). "A Critical Study of Selected Classification Algorithms for Liver Disease Diagnosis." International Journal of Database Management Systems (IJDMS) 3(2).

Rokach, L. and O. Maimon (2005). "Top-down induction of decision trees classifiers - a survey " IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 35(4): 476 - 487.

Safavian, S. R. and D. Landgrebe (1991). "A Survey of Decision Tree Classifier Methodology." IEEE Transactions on Systems, Man, and Cybernetics 21(3): 660-674.

Satinder P. Singh, Tommi Jaakkola, et al. (1995). "Reinforcement Learning with Soft State Aggregation." Advances in Neural Information Processing Systems 7 7.

Schilit, B. N., N. Adams, et al. (1994). "Context-Aware Computing Applications." IEEE Workshop on Mobile Computing Systems and Applications, December 8-9.

Schmidt, A. (2000). "Implicit Human Computer Interaction Through Context." Personal and Ubiquitous Computing, Springer.

Schmidt, A. and K. V. Laerhoven (2001). "How to Build Smart Appliances?" IEEE Personal Communications.

Shannon, C. E. (1948). "A Mathematical Theory of Communication." The Bell System Technical Journal 27(July, October,): 379–423, 623–656.

Sharma, H., R. Kuvedu-Libla, et al. (2008). "ConFra: A Context Aware Human Machine Interface Framework for In-vehicle Infotainment Applications." Proceedings of the International MultiConference of Engineers and Computer Scientists I.

Sharon, M. E. (2010). Facebook Blog Retrieved august 24 2010, from http://blog.facebook.com/blog.php?post=418175202130.

Siewiorek, D., A. Smailagic, et al. (2003). SenSay: A Context-Aware Mobile Phone. Seventh IEEE International Symposium on Wearable Computers

Sigg, S., S. Haseloff, et al. (2010). "An Alignment Approach for Context rediction Tasks in UbiComp environments." IEEE Pervasive Computing.

Silva, A. D. S. e. (2008). "Alien revolt (2005-2007): a case study of the first location-based mobile game in Brazil." IEEE Technology and Society Magazine 27(1): 18-28.

Song, L., D. Kotz, et al. (2006). "Evaluating Next-Cell Predictors with Extensive Wi-Fi Mobility Data." IEEE transactions on mobile computing 5(12): 1633 - 1649

Sørensen, C.-F., M. Wu, et al. (2004). a Context-Aware Middleware for Applications in Mobile Ad Hoc Environments. Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing. Toronto, Ontario, Canada ACM International Conference Proceeding Series. 77: 107 - 110.

Stoke, P. (2003). Philosophy. 100 Essential Thinkers Enchanted Lion Books.

Storey, V. C., V. Sugumaran, et al. (2004). "The Role of User Profiles in Context-Aware Query Processing for the Semantic Web " Lecture Notes in Computer Science. Natural Language Processing and Information Systems. 3136/2004(DOI: 10.1007/978-3-540-27779-8_5): 45-62.

Thomas Hofer, Wieland Schwinger, et al. (2002). Context-awareness on mobile devices – the hydrogen approach. Proceedings of the 36th Annual Hawaii International Conference on System Sciences.

Thomas, S. (2011). "Smartphone-driven, context-aware computing the next battle-ground." Retrieved May 5th 2012 2012, from http://memeburn.com/2011/10/its-all-about-context-computing-gartner/.

Thomason, R. H. (2008). Contextual intensional logic: type-theoretic and dynamic considerations. Perspectives on contexts,. L. S. Paolo Bouquet, Richmond H.Thomason, CSL Publications.: 189-215.

Wang, X. H., T. Gu, et al. (2004). Ontology based context modeling and reasoning using OWL. Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference.

Yi, J., F. Maghoul, et al. (2008). Deciphering Mobile Search Patterns: A Study of Yahoo! Mobile Search Queries. WWW 2008 / Refereed Track: Mobility. Beijing, China.

Yndurain, E., D. Bernhardt, et al. (2012). "Augmenting Mobile Search Engines to Leverage Context Awareness." IEEE Internet Computing 16 no. 2(March-April 2012): 17-25.

Yndurain, E., C. Feijoo, et al. (2010). "Context-aware mobile applications: implications and challenges for a new industry." ITP Journal 4 (Part 4): 9-20.

i https://www.facebook.com/about/location

ii https://foursquare.com/

iii http://gowalla.com/ (it has now merged with Facebook)